Calhoun: The NPS Institutional Archive

DSpace Repository

Theses and Dissertations | 1. Thesis and Dissertation Collection, all items

1997-09

Channel CAT: a tactical link analysis tool

Coleman, Michael Glenn

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/8075

This publication is a work of the U.S. Government as defined in Title 17, United States Code, Section 101. Copyright protection is not available for this work in the United States.

Downloaded from NPS Archive: Calhoun

http://www.nps.edu/library

Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

NPS ARCHIVE
1997.06
COLEMAN, M.

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

CHANNEL CAT :
A TACTICAL LINK ANALYSIS TOOL

by

Michael Glenn Coleman

September 1997

Thesis Advisor: Luqi

Approved for public release; distribution is unlimited

Thesis
C53434

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93943-5101

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93943-5101

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE September 1997	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE CHANNEL CAT: A TACTICAL LINK ANALYSIS TOOL	5. FUNDING NUMBERS
6. AUTHOR(S) Coleman, Michael G.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Marine Corps Tactical Systems Support Activity Camp Pendleton, California	10. SPONSORING/MONITORING AGENCY REPORT NUMBER N/A

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

The Tri-Service Tactical (TRI-TAC) standards for tactical data links mandate a terminal data rate of 32,000 bits per second. As greater demands for data throughput are placed upon tactical networks, it will become imperative that the design of future client/server architectures do not exceed the capacity of the TRI-TAC networks. This thesis produced an analysis tool, the Channel Capacity Analysis Tool (Channel CAT), designed to provide an automated tool for the anlysis of design decisions in developing client-server software.

The analysis tool, built using the Computer Aided Prototyping System (CAPS), provides designers the ability to input TRI-TAC channel parameters and view the results of the simulated channel traffic in graphical format. The size of data, period of transmission, and channel transmission rate can be set by the user, with the results displayed as a percent utilization of the maximum capacity of the channel.

.Designed using fielded equipment specifications, the details of the network mechanisms closely simulate the behavior of the actual tactical links. Testing has shown Channel CAT to be stable and accurate. As a result of this effort, Channel CAT provides software engineers an ability to test design decisions for client-server software in a rapid, low-cost manner.

14. SUBJECT TERMS United States Marine Corps, command, control, network, time division multiplex, links, analysis, modeling, prototyping.	15. NUMBER OF PAGES 101
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

Approved for public release; distribution is unlimited

CHANNEL CAT :
A TACTICAL LINK ANALYSIS TOOL

Michael Glenn Coleman
Captain, United States Marine Corps
B.S., United States Naval Academy, 1988

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1997

NPS Archive
1997.09
Coleman, M.

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93943-5101

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

ABSTRACT

The Tri-Service Tactical (TRI-TAC) standards for tactical data links mandate a terminal data rate of 32,000 bits per second. As greater demands for data throughput are placed upon tactical networks, it will become imperative that the design of future client/server architectures do not exceed the capacity of the TRI-TAC networks. This thesis produced an analysis tool, the Channel Capacity Analysis Tool (Channel CAT), designed to provide an automated tool for the anlysis of design decisions in developing client-server software.

The analysis tool, built using the Computer Aided Prototyping System (CAPS), provides designers the ability to input TRI-TAC channel parameters and view the results of the simulated channel traffic in graphical format. The size of data, period of transmission, and channel transmission rate can be set by the user, with the results displayed as a percent utilization of the maximum capacity of the channel.

Designed using fielded equipment specifications, the details of the network mechanisms closely simulate the behavior of the actual tactical links. Testing has shown Channel CAT to be stable and accurate. As a result of this effort, Channel CAT provides software engineers an ability to test design decisions for client-server software in a rapid, low-cost manner.

TABLE OF CONTENTS

I. INTRODUCTION 1

A. BACKGROUND 1

B. MOTIVATION 1

C. SUMMARY OF CHAPTERS 4

II. THE TRI-SERVICE TACTICAL NETWORK 5

A. BACKGROUND 5

1. Time Division Multiplexing 5

2. Necessity 5

3. Solution 7

B. SPECIFICATIONS 7

1. Data Rate 8

2. Error Rates 8

C. CHAPTER SUMMARY 9

III. CHANNEL CAT DESIGN AND USE 11

A. COMPUTER AIDED PROTOTYPING SYSTEM 11

1. Overview 11

2. Prototyping Process 11

3. Prototype System Description Language 12

a. Operators 12

b. Streams 12

c. Types 13

d. Timing Constraints 13

e. Triggers 13

f. Execution Guards 14

g. State Variables 14

B. ASSUMPTIONS 14

C. SYSTEM DECOMPOSITION 15

1. Parameters 16
a. Constant 16
b. Run Time 16
2. User Interface 17
a. Input Panel 17
b. Results Panel 21
3.. TRI-TAC Link Operators 23
a. Transmitter 25
b. Receiver 27
c. Channel 29
d. Utilization Calculator 30
D. OPERATION 31
1. User Input 31
2. Results 31
a. Text-based 31
b. Graphics-based 32
E. CHAPTER SUMMARY 32
IV. TESTING AND VALIDATION 35
A. TESTING 35
1. Plan 35
2. Criteria 35
B. RESULTS 35
1. Accuracy 36
2. Boundary Conditions 36
3. Fidelity 36
C. CHAPTER SUMMARY 38
V. CONCLUSIONS 39
A.. STRENGTHS OF CHANNEL CAT 39

B. FUTURE WORK 40

LIST OF REFERENCES 43
BIBLIOGRAPHY 45
APPENDIX A PSDL SOURCE CODE 47
APPENDIX B ADA SOURCE CODE 51
APPENDIX C USER INTERFACE CODE 69
APPENDIX D TESTING/VALIDATION CRITERIA AND RESULTS 77
INITIAL DISTRIBUTION LIST 85

LIST OF FIGURES

Figure 1. Upward TDM Example 6

Figure 2. Top-level PSDL Graph of Channel CAT 18

Figure 3. Channel CAT Input Panel 19

Figure 4. Channel CAT Results Panel 22

Figure 5. Results of Assumptions 23

Figure 6. Concept Drawing 24

Figure 7. PSDL Graph of TRITAC_LINK Operator 26

LIST OF TABLES

Table 1. Range of Acceptable Values for Channel CAT Input Panel Parameters.....20

I. INTRODUCTION

A. BACKGROUND

Military communications are in the midst of a major revolution. Digital equipment and transmission mediums have completely permeated the Department of Defense's (DoD) command and control (C2) infrastructure. This new capability has enhanced the warfighting ability of all services by providing more reliable means of transmitting digital data. As the military C2 systems migrate from stand alone systems with little or no interoperability to a cohesive network of client-server systems employed in a common operating environment (COE), greater demands will be placed on the Tri-Service Tactical Network (TRI-TAC). It is an established fact that any distributed system, particularly client-server designs, experience degraded performance as a result of distributed query processing. [ELMA94].

B. MOTIVATION

Client-server architectures are a mature technology and place large bandwidth requirements on networks. The intelligent and prudent design of client-server processes is paramount if real-time processing demands are to be supported by the TRI-TAC backbone. Real-time processing and sharing of data is necessary to ensure accurate and timely decision making by military commanders. The Persian Gulf War exemplified the power of being able to collect, process, and act upon information. If TRI-TAC follows the historical lifecycle for military systems, TRI-TAC will remain as the primary network

system for many years into the future. Due to the wide-spread use of TRI-TAC within DoD and budget constraints which will inhibit major restructuring of DoD tactical data systems, the DoD must focus on the assumption that the network currently employed is our network for the near future.

The future of TRI-TAC as the backbone of the Marine Air Command and Control System (MACCS) is of specific importance to this thesis. The MACCS is the collection of commanders, systems, and weapon platforms primarily responsible for execution of all aspects of the air battle for the United States Marine Corps (USMC). Heavy data traffic and large data files are normal, existing in the form of electronic mail, file transfers, intelligence data, operational orders and documents, facsimile traffic, radar data, database operations, logistics information, military messages, and digital telephone connections. The main means of transmitting these many and varied types of information is through the use of TRI-TAC links, interconnected with various sizes of digital switches.

It is, therefore, absolutely imperative that the limited bandwidth available on the TRI-TAC network be treated as a precious resource. The client-server systems to be fielded must employ schemes which minimize network traffic. Gone are the days when military systems can built under the assumption that it is acceptable to transmit large pieces of data between systems when only a portion of the data is needed. Systems designers must either guess at TRI-TAC channel capabilities, spend inordinate amounts of time computing data to predict the tactical network behavior, or ignore the reality of limited bandwidth and simply design without regard for the TRI-TAC network

capabilities. Obviously, guessing at and ignoring problems are not optimal methods for system design. The only acceptable alternative for systems designers is to make complicated calculations to measure every possible set of parameters. A robust analysis tool can reduce the time and effort involved in making such calculations and can do so with guaranteed accuracy.

The designer must consider four system parameters. The TRI-TAC channel speed is constant in theory, with an upper limit of 32,000 bits per second [MAWT93], but experience has shown this maximum speed to be often unattainable because of such factors as terrain, vegetation, and weather. Besides the network maximum speed, the three other factors which influence a TRI-TAC channel's capacity are the size of the data being passed over the channel, the frequency of data being added to the channel, and the distance between the logical processes (computers).

Automated tools are needed to support the design of client-server software for TRI-TAC systems. Channel Capacity Analysis Tool (Channel CAT) was built to simplify the complex variety of possible combinations of channel speed, data size, period of transmission, and distance. Channel CAT, detailed within this thesis, was designed and built using the United States Naval Postgraduate School's (NPS) Computer Aided Prototyping System (CAPS). Channel CAT is a model of a single channel on a TRI-TAC link, allowing the user to vary the data and channel parameters. This tool allows a client-server module designer the ability to test the feasibility of their proposed client-server module on an accurately simulated TRI-TAC channel.

The ability to test design decisions on an accurate model is important in identifying design flaws before the system is fully developed. Because the cost to fix errors increases as the system is developed further, it is both sound engineering and responsible cost control to minimize the introduction of errors and identify errors as early as possible in the development cycle.[BERZ91]

C. SUMMARY OF CHAPTERS

Chapter II provides the pertinent details of the TRI-TAC network. The CAPS version of Channel CAT is explained in Chapter III. Design decisions are included, as well as specific information on system decomposition. The testing and validation of Channel CAT are contained in Chapter IV. Chapter V provides the author's conclusions on this effort. Three appendices are included. Appendix A contains the Prototype System Description Language (PSDL) source code for Channel CAT. Appendix B contains both the Ada source code which was translated from the PSDL code and all Ada packages written by the author. Appendix C contains the Ada source code for the user interface. Appendix D contains the test/validation criteria and results for Channel CAT.

II. THE TRI-SERVICE TACTICAL NETWORK

A. BACKGROUND

In software engineering, one of the most important steps in providing a software solution is developing a keen understanding of the real-life issues of the actual problem. This section provides a cursory explanation of basic time division multiplexing (TDM), the necessity of using such technology within USMC command and control architectures, and the basic solution which has been adopted. This information is provided for the readers with no knowledge of USMC tactical communication needs.

1. Time Division Multiplexing

Multiplexing is an established method for utilizing a single transmission path for several concurrent connections. Time division multiplexing accomplishes this service by providing each connection a periodic slice of time during which that connection has sole use of the entire transmission path. Each connection is part of a circular queue and is provided sole use of the transmission path during its assigned time slice. This variation of multiplexing is generally known as upward multiplexing [TANE96]. Figure 1 demonstrates how this multiplexing scheme can be used by two connections to share a single transmission path.

2. Necessity

The attractive nature of using any multiplexing scheme stems from the ability to use a single transmission path as a means of providing multiple connections. This reduces

the number of physical connections which need to be made in support of command and

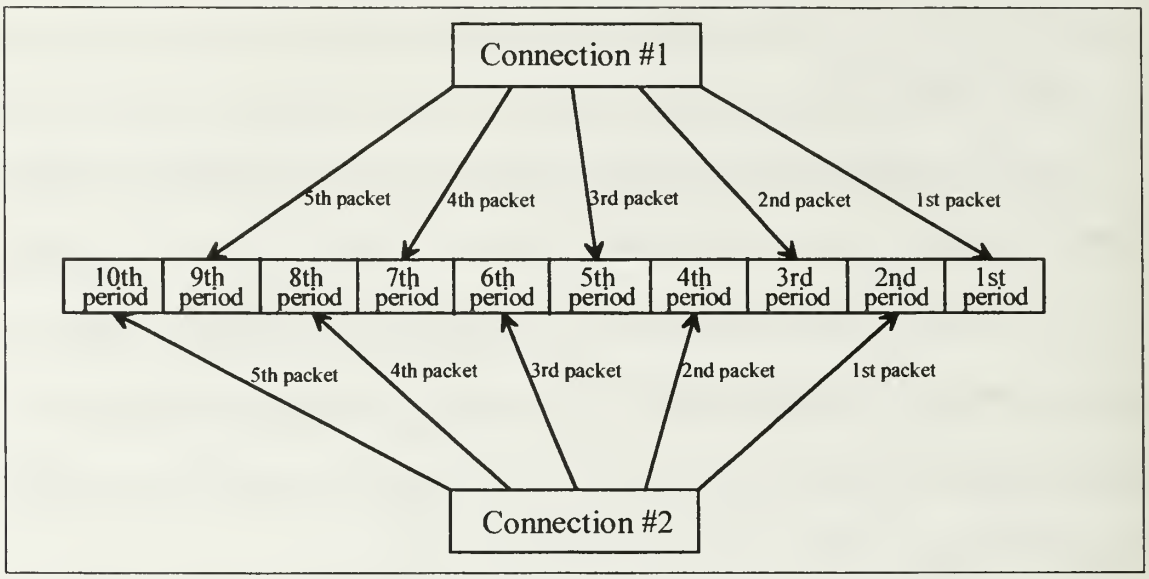

Figure 1. Upward TDM Example

control systems. The resulting benefits include reduced equipment costs, greater mobility through less bulk of equipment, fewer links for communications personnel to install, operate, and maintain, and greater flexibility and response to changing battlespace operational requirements. As the need to move digital data within the battlefield increases, multiplexing systems have become vital in providing a backbone for the command and control system.

Every facet of the Marine Corps command and control structure, including administration, intelligence, operations, logistics, planning, and communications, has some reliance on information technology. When properly designed and managed, information technology can be a potent force multiplier.

3. Solution

The United States Marine Corps used analog Frequency Division Multiplexing systems in earlier data transmission systems. Such systems are best used only for analog data such as voice communications. The introduction of digital data from computers, faxes, and other terminal equipment throughout all the armed services led to a joint effort to establish a set of standards for digital TDM and interoperability between the services. The network which sprang forth from this effort is now historically known as the Tri-Service Tactical Network (TRI-TAC).

TRI-TAC equipment has been fielded and is used in Marine Corps units ranging in size from Marine Expeditionary Forces commanded by either 2 or 3 star generals down to operational battalions commanded by Lieutenant Colonels. TRI-TAC equipment is ruggedized, weather resistant, and easily transported. An obvious premium has been placed on flexibility and reliability of the systems as part of a command control system. The ability to utilize digital signals for the transmission of digital data has caused sweeping changes in the Marine Corps command and control systems. More information of greater accuracy can be assimilated in a shorter period of time, greatly reducing the time of the decision cycle of commanders.

B. SPECIFICATIONS

TRI-TAC standards are specified in elaborate detail.[MAIN91] For the purpose of this discussion, the pertinent details are those concerning network data rates and error rates.

1. Data Rate

TRI-TAC standards prescribe a data rate of 32,000 bits per second (bps) as the maximum achievable speed for each channel in the multiplexed link. The standards do address the use of much greater and much lesser speeds, but 32,000 bps was established as the inter-service speed as a means of ensuring a level of interoperability.

By today's standards, 32,000 bps is not fast. The average household telephone line can support data rates up to 64,000 bps. Emerging network technologies are utilizing network speeds at 1 gigabit per second, rougly 30,000 times faster than TRI-TAC. It should be remembered that this standard is meant to span commands from the national and allied level to front-line units, where hostile fire and weather conditions often prohibit the use of delicate network equipment needed to achieve high network speeds.

2. Error Rates

Any military command and control system is subject to degradation from many sources. Some degradation is caused by environmental factors, such as rain, snow, sand storms, vegetation, and solar flares. Other problems result from rugged use of the equipment. Finally, software errors can result in decreased throughput. In military communications, any link which achieves a bit error rate not exceeding 1 error per 10 million bits is of good quality. This does not mean that all military links are of such good quality. Often, the operational demands require command and control systems be employed in a less than ideal manner, thereby resulting in partially degraded systems. As the bit error rates rise from such factors, error correcting protocols initiate retransmission

of lost data. The realized throughput of the links decreases, impacting the speed of the command and control cycle. The software designer of client-server modules for the military environment must consider such degradation of channel speed as normal. What results from such a conclusion is a lack of a guaranteed speed which will always be available for the links. The software designer now must be heavily concerned with minimizing the amount of bandwidth used by client-server modules.

C. CHAPTER SUMMARY

TDM has provided military organizations the capability to move digital information around the battlefield. The United States Marine Corps, in conjunction with the other armed services, now uses the TRI-TAC network system to support command and control architectures. TRI-TAC standards call for a data rate of 32,000 bps, but this often proves to be unattainable due to various factors. The software designer is faced with developing usable client-server modules which both accomplish the desired tasks and minimize the bandwidth used while doing so.

III. CHANNEL CAT DESIGN AND USE

A. COMPUTER AIDED PROTOTYPING SYSTEM

1. Overview

Channel CAT was designed and built using the Computer Aided Prototyping System (CAPS). CAPS was developed at the United States Naval Postgraduate School (NPS) Computer Science Department by the CAPS development team. Designed as a Computer Aided Software Engineering (CASE) tool, CAPS is used to build prototypes of real-time systems. CAPS is a complete development environment, incorporating graphical system decomposition and design, interface design and integration, real-time scheduling, feasibility checking and enforcement, compiler support, and reuse support via a software base. All the support is accessed through a single user interface.

2. Prototyping Process

Prototyping is widely recognized as an extremely useful method for refining requirements of proposed systems. The prototyping process is iterative. Initial requirements are used to design a prototype system, which is demonstrated to the customer. Based on the performance of the prototype, the customer validates the prototype. If considered valid, construction of a production system may proceed. However, if the user has found the performance of the prototype to be unsuitable, the problems can be addressed by refining the requirements used to build the prototype. The cycle continues until the customer has been provided a prototype which is considered by

them to be valid. With proper software tools, the prototyping cycle can be completed quite rapidly and at a cost below building actual systems. As discussed earlier, many of the errors resident in a fielded system are introduced at the requirements analysis stage. Prototyping cycles can be used to eliminate many errors in the requirements analysis.

3. Prototype System Description Language

This section provides background information and details on several aspects of the CAPS design representation that were used in the design of Channel CAT. [LUQI97]

a. Operators

Operators are drawn by the designer in the CAPS graphical editor as either circles or rectangles. Rectangles represent simulations of external systems. Circles represent the proposed software components. Each operator is assigned a unique name and can be provided a Maximum Execution Time (MET) from within the graphical editor. Operators can be decomposed by the designer. Such operators are shown as double circles and are called composite operators. Any operator which is not decomposed is called an atomic operator and will be eventually implemented in the Ada programming language. Operators which output a value based solely on a set of input values are called functions. An operator whose output is completely or partially based on one or more state variables is called a state machine.

b. Streams

Streams represent communication and are used to connect operators. Two types of streams are possible. Sampled streams are those streams which act as the

equivalent of a programming variable. Data flow streams are any streams which have a consuming operator which "fires" on every occurrence of data on the stream and removes each occurrence after it is read.

c. Types

CAPS has robust facilities for the management of abstract data types defined by the user. All such types can be implemented in Ada.

d. Timing Constraints

Timing is the major obstacle in understanding and modeling a proposed real-time software system. CAPS provides strong mechanisms for the establishment and enforcement of timing constraints. Any operators which are given timing constraints are considered as time critical and are given scripted, static scheduling priority by CAPS. Certain timing constraints can be used to modify the behavior of an operator, based on whether the operator is to be a periodic or sporadic operator. Periodic operators are assigned a Maximum Execution Time (MET) and a Period (P). They can also be assigned a Finish Within (FW) time constraint. The MET is the block of CPU time to be scheduled for the execution of the operator. The Period is used to control how often the operator executes. FW is used to enforce completion of execution within all or some of the period.

e. Triggers

CAPS has two types of triggers, which restrict the conditions under which an operator can fire. The "BY ALL" triggers can be assigned to an operator when it is desired that when, for every stream listed in the triggering set, new data is present. This

does not necessarily mean that every stream entering the operator is listed in the triggering set, but such an arrangement is possible. The second type of triggering, "BY SOME", is similar to "BY ALL" except that new data need only arrive on at least one of the streams listed in the triggering set to fire the related operator.

f. Execution Guards

Execution guards are another means of regulating the execution of an operator. These guards are conditional statements which are evaluated based on data received from any one or more of the streams or state variables. In the instance where the execution guard condition is not satisfied, the operator does not execute, but the data present on the streams is still consumed.

g. State Variables

State variables are important for ensuring the CAPS graph of the prototyped system is a directed acyclic graph (DAG), as any cycle will prevent the static scheduler in CAPS from finding a feasible schedule. This mandates there be no possible cycles, intentional or otherwise. If such a cycle is present, a state variable can be declared in the editor, using the same name as one of the streams associated with the cycle. CAPS then considers the cycle broken, essentially removing the stream from the graph for scheduling purposes.

B. ASSUMPTIONS

The development of Channel CAT required a clear understanding of the USMC TRI-TAC network system and knowledge of which aspects of the TRI-TAC system are

pertinent to client-server software design. One of the most clarifying assumptions the author made was to recognize the independence of format and content from channel throughput. Such a decision is not totally realistic, as certain data associated with certain applications may produce better or worse throughput. The assumption of performance independence from format and content was made to provide a more pure measurement of channel performance by ignoring any enhancements or faults in the actual software applications and to maintain the focus of Channel CAT on the physical layer of the TRI-TAC network.

TDM links mandate a fixed maximum packet size. This activity was not modeled. The disassembly and reassembly of packets can clearly be seen as having no significant effect on the actual movement of data elements over a channel, particularly when the limiting factor in channel performance is the channel's data rate.

The final choice which was made was to model the TRI-TAC channel in a simplex mode only. Channel CAT's goal is to provide a tool for measuring the ability of a TRI-TAC channel to push a certain data size, at a certain period, at a certain speed. The activities associated with duplex channel operations are heavily dependent on the behavior of the parent software applications and are, for this reason, ignored in Channel CAT.

C. SYSTEM DECOMPOSITION

This section details the elements of an actual TRI-TAC TDM channel which were incorporated into Channel CAT. This is not meant to be an inclusive description of TDM, but it meant to give perspective on certain important elements within the TRI-TAC

network and how they were translated into actual implementation within CAPS.

1. Parameters

a. Constant

As discussed earlier, the maximum channel speed of a TRI-TAC channel is standardized at 32,000 bps, but this has also been shown to be an upper limit which is rarely achieved. In Channel CAT, the user may set the channel speed from 1 to 4000 bytes per second (32,000 bits per second).

The granularity of Channel CAT is focused toward providing a usable interface. In support of such an aim, the manipulation of the modeled channel occurs at 250 ms intervals. Greater precision is possible, but the author contends more precision does not enhance the value of Channel CAT to the user.

b. Run Time

As discussed in Chapter 1, the parameters which the author incorporated into Channel CAT are the data size, the period at which the data is transmitted to the channel, the distance of the link, and the rate at which the data is physically transmitted across the channel.

The distance between nodes factor was deleted from the final version of Channel CAT because of its relative insignificance. In the channel being used to communicate 1000 bytes of information/data every second, with the channel running at 4000 bytes per second, the entire data portion can be handled in 250 milliseconds. If we further suppose the physical speed of the transmission medium is 2/3 the speed of light

(200000 kilometers per second) the relatively small impact of internodal distance is apparent. Sending the 1000 bytes over a 2 kilometer distance introduces an additional delay of .01 milliseconds, resulting in a total transmission time of 250.01 milliseconds. Sending the same data over a distance of 200 kilometers, the delay incurred is only 1 millisecond and the total transmissiontime is now 251 milliseconds. Because the relative increase in these two cases is only .0004% and .004% of the original total time, the distance factor was deemed unimportant and was not incorporated into the final version. Figure 2 is the top-level PSDL decomposition of Channel CAT. Two ,operators USER_INTERFACE AND TRITAC_LINK, and five streams, DATA_RATE, DATA_PERIOD, DATA_SIZE, START_STOP, and UTILIZATION, are shown in the graph.

2. User Interface

The user interface was divided into two windows, an interactive input panel and a results panel. The results panel is not interactive, merely providing the user the ability to monitor the modeled channel.

a. Input Panel

The user_interface operator is not decomposed any further within the CAPS graphical editor. The implementation of user_interface is accomplished using TAE+ version 5.3, which is integrated into CAPS. The Ada code used to generate the user interface is included as Appendix C. The author chose to implement a two panel (window) interface. The first panel , Figure 3, is used strictly for input of parameters and

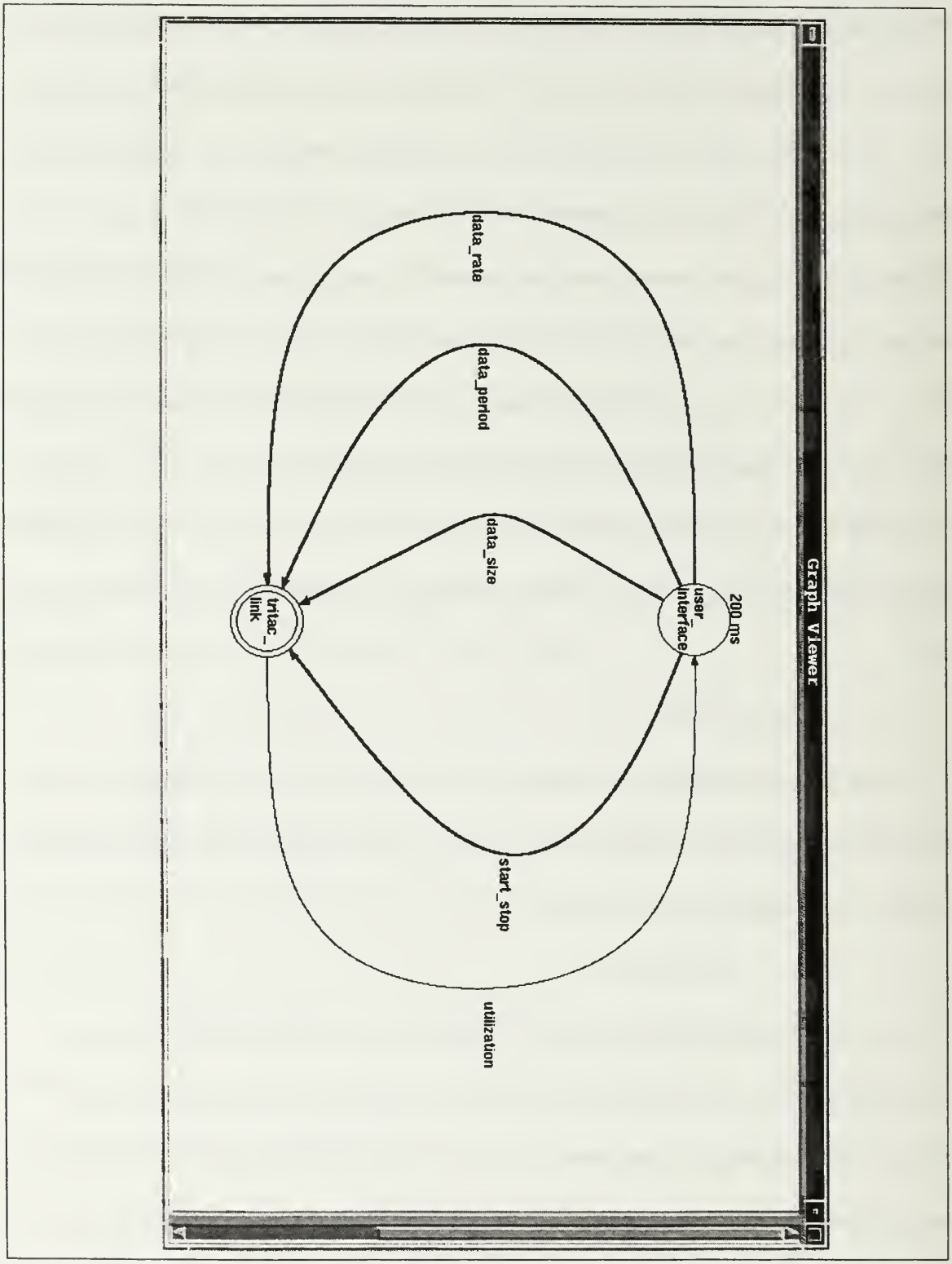

Figure 2. Top-level PSDL Graph of Channel CAT

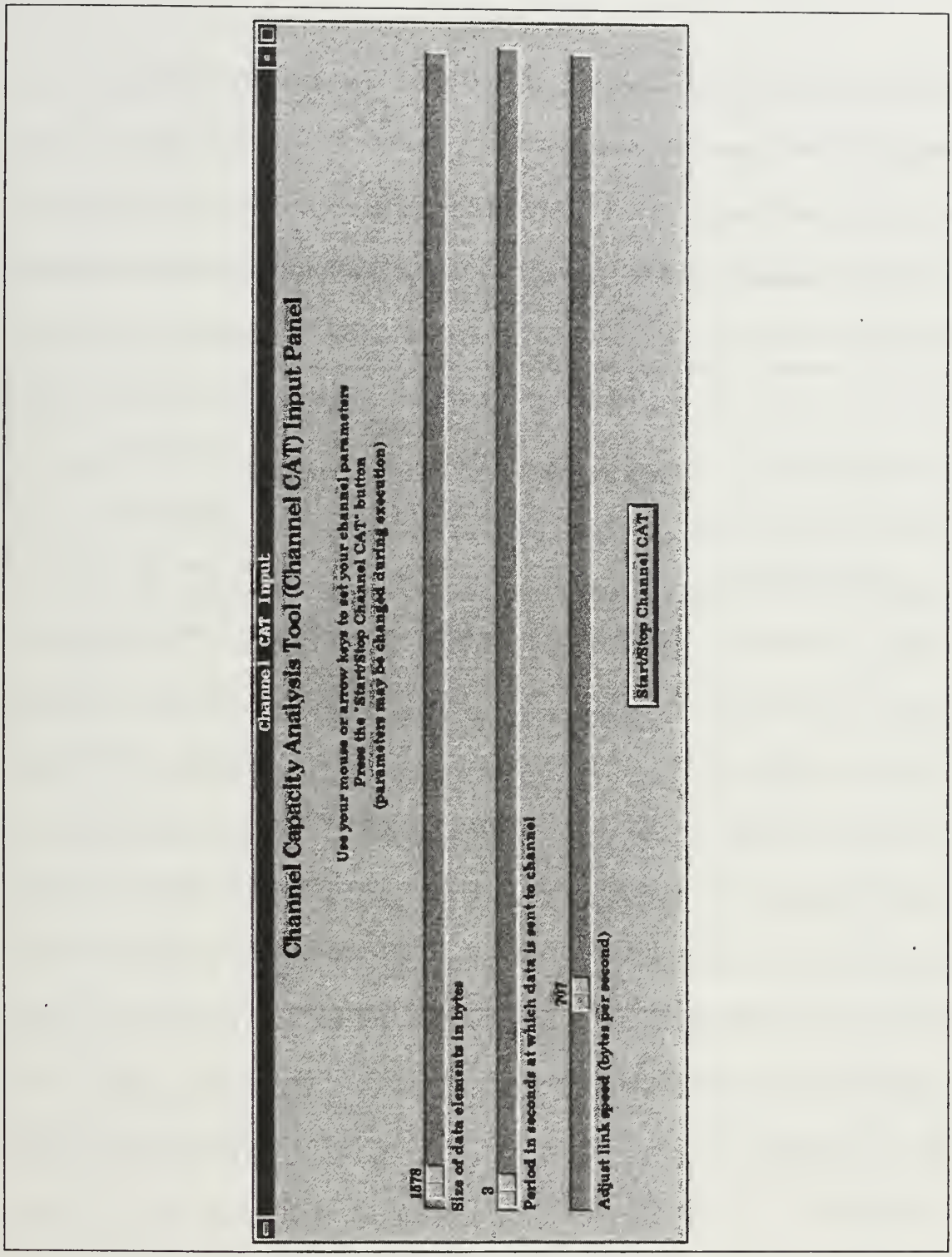

Figure 3. Channel CAT Input Panel

for starting and stopping the operation of the channel. Figure 3 is a screen capture of Channel CAT's input panel during execution. As can be seen in Figure 3, the user has set the channel parameters to a data size of 1,578 bytes, a period of transmission of 3 seconds, and a channel speed of 707 bytes per second. The acceptable range of value for each parameter is shown in Table 1. The user of Channel CAT is allowed to modify the

	Minimum accepted value	**Maximum accepted value**
Data size	1 byte	250,000 bytes
Data period	1 second	1800 seconds
Channel (link) rate	1 byte per second	4000 bytes per second

Table 1. Range of Acceptable Values for Channel CAT Input Panel Parameters

values for the parameters at any time, even during run-time of the prototype. When the prototype is initially loaded, all parameters are set to their minimum value and the prototype is in the paused mode. The prototype is toggled between the running mode and the paused mode by the use of the 'Start/Stop' button located at the bottom of the input panel. This button passes a boolean value to the prototype, which causes most of the operators to cease execution. These operators have been given an execution guard which checks this boolean value at the beginning of every attempted execution. Lines 275 and 421 of Appendix B are the associated execution guards (execution trigger condition checks) for the RCVR and XMTR operators.

b. Results Panel

The second panel of the user interface is the results panel shown in Figure 4. This particular panel is completely non-interactive with the user. The major functionality of the results panel is supplied by a Dynamic Data Object called a stripchart. Similar in general behavior to medical devices used to measure heart activity, the results graph's horizontal axis is a 30 second segment of time and its vertical axis is the percentage of the maximum channel capacity used. The current value is displayed numerically in the right hand corner and graphically at the rightmost edge of the graph. Three colors are used to draw the utilization graph: green for values below 75 %, yellow for values from 75 to 99 %, and red to draw the graph for all values of 100 % or greater. These colors are used in a manner analogous to traffic signals. The results panel continues to display a value if the prototype is stopped by the user. The resulting graph will simply be a horizontal line beginning at the last value supplied and will continue to display this value until the prototype is either selected to run by the user or the user exits Channel CAT. The effect of continued display of the last valid value on the results graph is intentional and reflects the internal state of Channel CAT when the user pauses Channel CAT. All internal values are maintained in their most recent state prior to pause being selected by the user, so the display of the last valid value for UTILIZATION during pause is considered by the author to be appropriate.

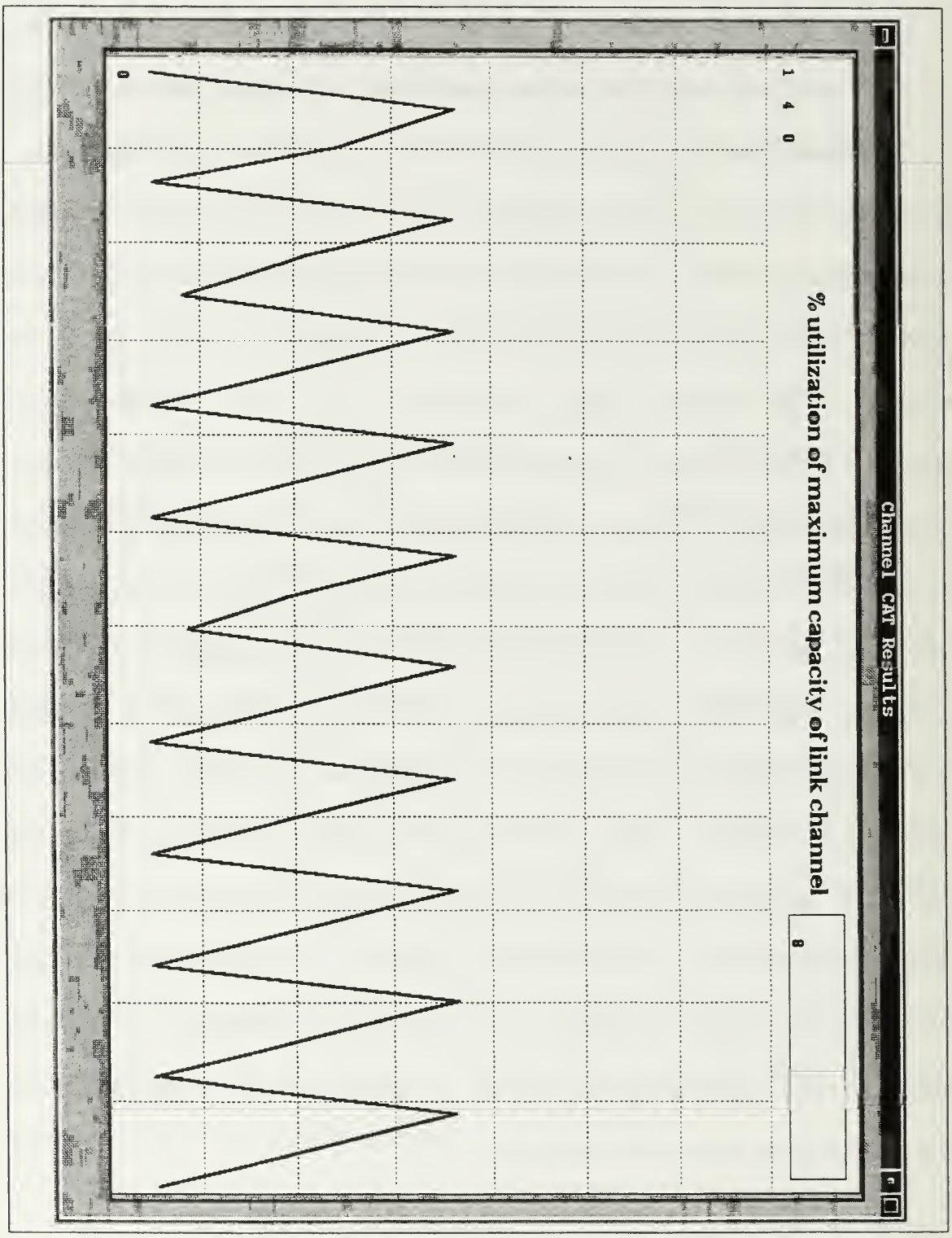

Figure 4. Channel CAT Results Panel

3. TRI-TAC Link Operators

The tritac_link operator was decomposed initially into three abstract entities: a transmitter, a receiver, and a medium over which to communicate, as shown in Figure 5.

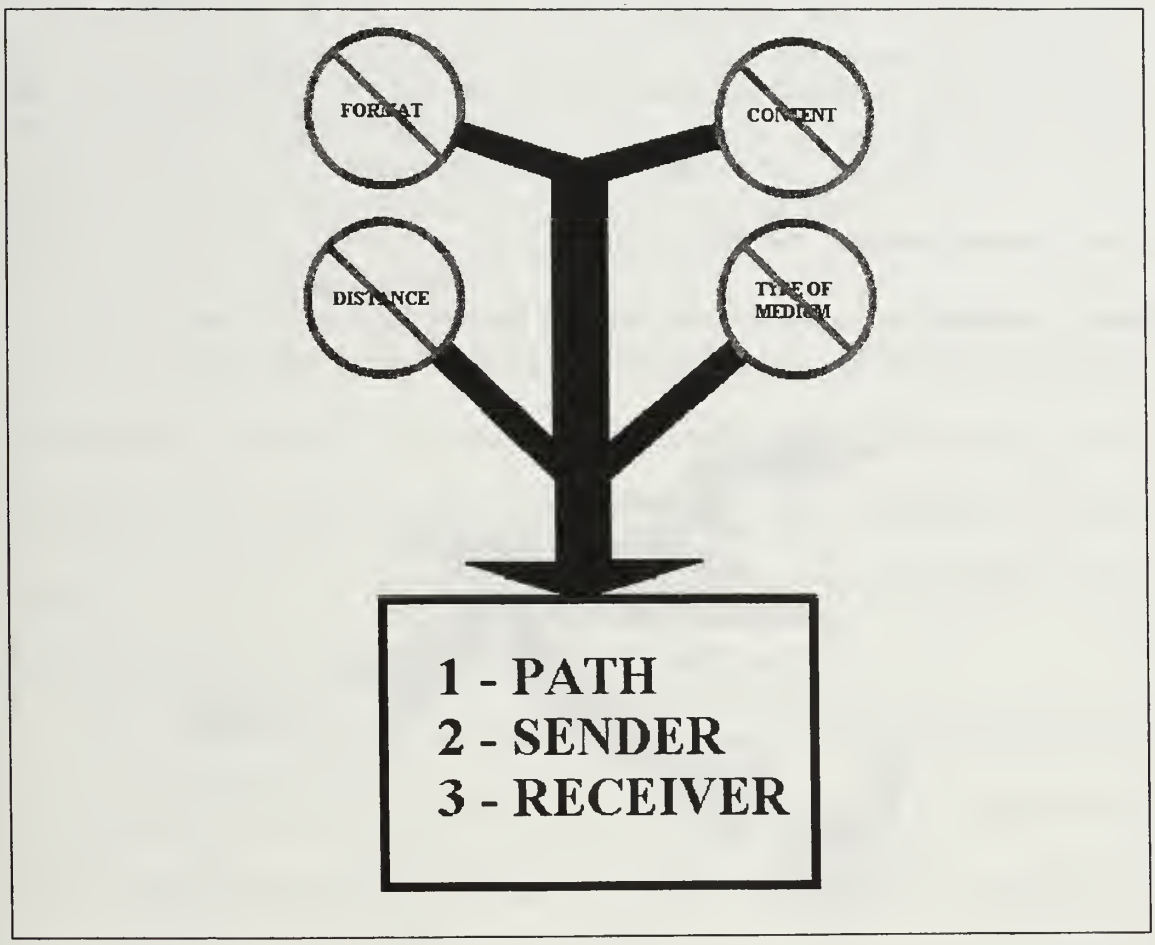

Figure 5. Results of Assumptions

Format, content, internodal distance, and type of medium were relatively insignificant and can be ignored for modelling purposesSuch abstraction of a TRI-TAC link strongly supports the application independence desired by the author. With this simple three entity model, the link has been trimmed down to the essential elements for measuring channel

capacity. A fourth operator was added to handle simple results formatting. The functionality of this operator is not time critical.

Figure 6 is the concept drawing for the behavior of the transmitter (SENDER), the receiver (RECEIVER), and the channel (IN/OUT BOX). The IN/OUT BOX is the abstracted channel. At appropriate times as set by the user, the SENDER places the set amount of data in the box. The receipt of the data is accomplished by the RECEIVER at a pre-determined interval. The RECEIVER takes all or a portion of the contents of the channel, depending on the network data rate, and discards the data as shown in Figure 6.

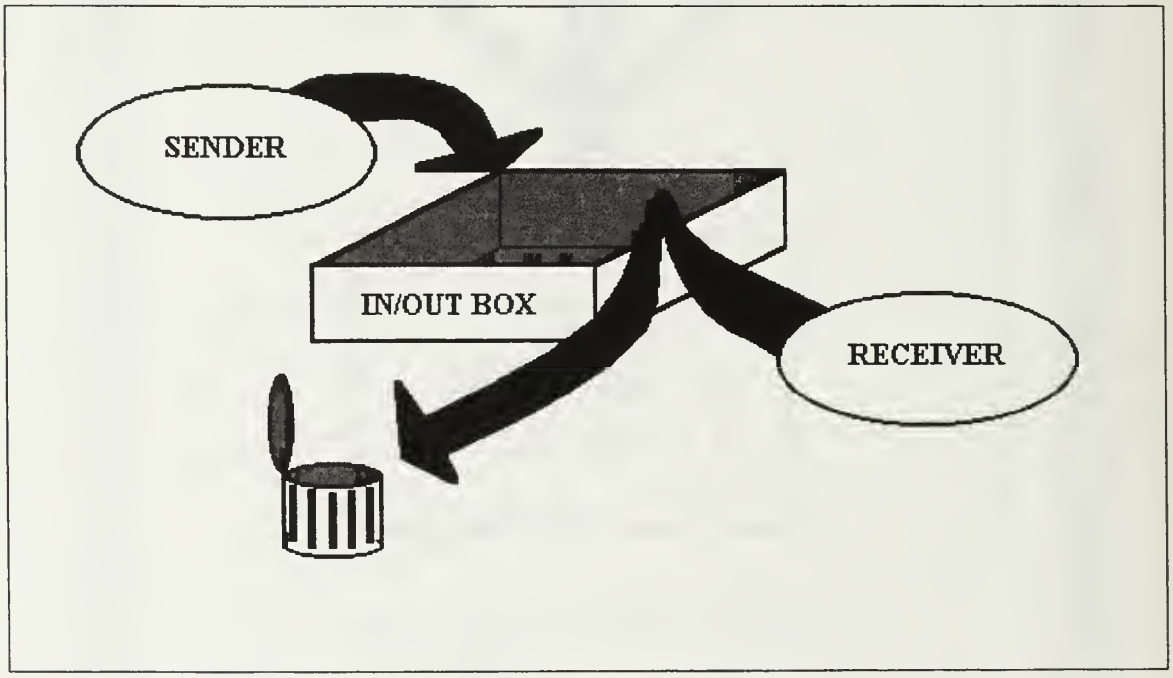

Figure 6. Concept Drawing

This analogy is accurate due to the earlier assumption that the format and content of the data are independent of the performance of the physical layer of a TRI-TAC channel and the data can be simply consumed or discarded by the RECEIVER. This abstraction of a

channel on a TRI-TAC link is the concept for implementation of the atomic CAPS operators XMTR, RCVR, and CHANNEL_BUFFER. The fourth atomic operator of the composite operator TRITAC_LINK is called UTILIZATION_CALCULATOR. As previously mentioned, there was a need to format the results of the modeled channel and UTILIZATION_CALCULATOR fulfills this need. Figure 7 is the PSDL graph of the decomposed TRITAC_LINK operator into the four atomic operators.

a. Transmitter

Lines 748 through 813 of Appendix B is the Ada code for the atomic implementation of XMTR. Five parameters are involved with XMTR. Three of the parameters, DATA_PERIOD, DATA_SIZE, PERIOD, and START_STOP, are 'in' mode parameters. The PERIOD_COUNTER parameter is an 'in out' mode parameter. Finally, BUFFER_MODIFICATION is an 'out' parameter.

XMTR was assigned a period of 1000 milliseconds during design. This was necessary so that XMTR would be statically scheduled by CAPS, ensuring execution priority. The conceptual design of Channel CAT allows for the user to manipulate the period at which data is sent to the channel. The necessity of assigning a period to XMTR runs counter to this design consideration. A method was needed to control the execution of XMTR to take of advantage of the XMTR operator having priority by being statically scheduled, but ensure the XMTR operator executes at the period desired by the user. A state variable, PERIOD_COUNTER, was introduced during design to control the execution of XMTR. Lines 71 and 74 of Appendix A show the PERIOD_COUNTER

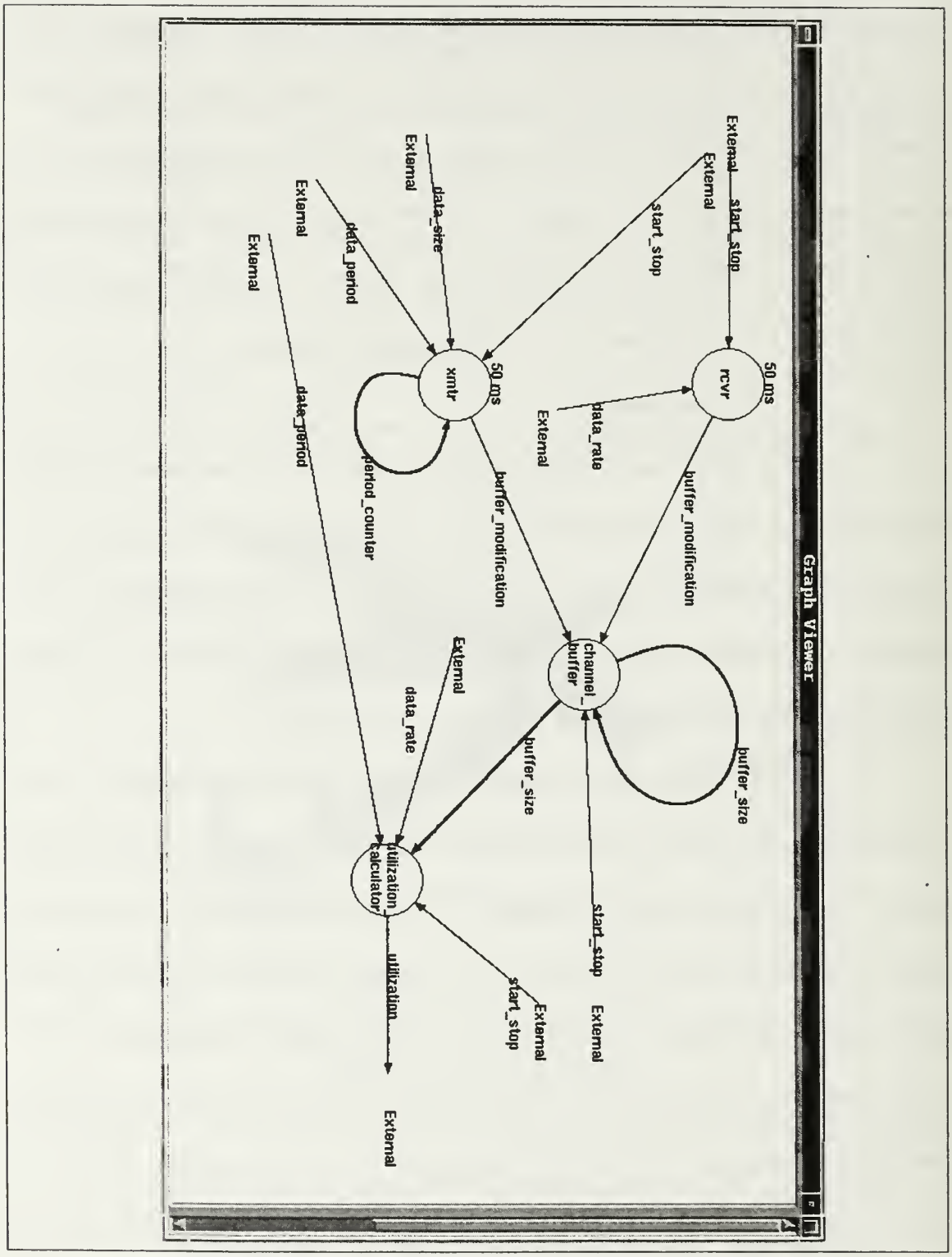

Figure 7. PSDL Graph of TRITAC_LINK Operator

stream declared as a state variable and initialized to a value of 0. Lines 793 through 812 is the body of the XMTR procedure. Line 800 shows the first instruction executed is to increment PERIOD_COUNTER. Line 806 then performs a condition check to compare the values of PERIOD_COUNTER and DATA_PERIOD. If the values are equal, XMTR assigned the value of DATA_SIZE to BUFFER_MODIFICATION. If the condition fails, BUFFER_MODIFICATION is given a value of 0. The analogy for this execution control is found in the familiar ticketing system used at bakeries and barber shops. Every customer takes a number upon entering the business (DATA_PERIOD). The customer is then forced to wait for service until their number matches the number displayed on the sign in the business (PERIOD_COUNTER).

As can be seen in Figure 7, the stream BUFFER_MODIFICATION appears twice, drawn from XMTR and RCVR operators to the CHANNEL_BUFFER operator. The BUFFER_MODIFICATION data values supplied by XMTR are always positive.

b. Receiver

Lines 635 through 687 of Appendix B is the Ada code for the atomic implementation of the RCVR operator. RCVR has three parameters. DATA_RATE and START_STOP are 'in' mode parameters. BUFFER_MODIFICATION is an 'out' parameter.

The RCVR procedure body is shown in lines 670 through 687. Two local variables are declared initially, ops_per_second and amount_per_500_ms. The RCVR

operator has a period of 500 milliseconds. A decision on the level of granularity was necessary as it is impractical to implement the RCVR operator in such a manner that it removes a single byte during every execution. Such behavior is not only unattainable within CAPS, it is also not discernible by the user. A period of 500 milliseconds was chosen through a series of trials, satisfying the desired granularity of less than 1000 milliseconds for the RCVR operator. Data can be sent to the channel at a period of no less than 1000 milliseconds, so it was necessary to implement a RCVR period of less than 1000 milliseconds as a means of capturing the behavior of the channel at this lower bound. Three different values for the RCVR's period were tested. The first value, 100 milliseconds, led to numerous timing errors and caused the results panel's graph to have gaps. A period of 250 milliseconds was tested and was found to alleviate the timing errors, which also improved the quality of the results graph. 500 milliseconds was tested and was found to eliminate nearly all timing errors and yielded a very smooth behavior on the results graph.

When RCVR executes every 500 milliseconds, the DATA_RATE value is compared to a value of 2. If greater than 2, lines 680 and 681 assign amount_per_500_ms the appropriate value based on the DATA_RATE and ops_per_second. BUFFER_MODIFICATION is then assigned the negative value of amount_per_500_ms. If DATA_RATE was equal or less than 2, BUFFER_MODIFICATION is assigned the minimum possible value of -1 byte. This operation was found necessary as the integer rounding which occurs in line 680 and 681 was yielding a value of 0 for DATA_RATE

values less 2.

The RCVR procedure always generates a negative value for the BUFFER_MODIFICATION parameter.

c. Channel

CHANNEL_BUFFER is the portion of Channel CAT which models the physical TRI-TAC connection. Lines 584 through 634 in Appendix B contains the Ada code for the implementation of CHANNEL_BUFFER. The CHANNEL_BUFFER procedure has three parameters. BUFFER_MODIFICATION and START_STOP are both 'in' parameters. BUFFER_SIZE is an 'in out' parameter.

Lines 619 through 634 of Appendix B is the procedure body of CHANNEL_BUFFER. The normal execution of CHANNEL_BUFFER can be seen in line 626, where BUFFER_MODIFICATION is added to BUFFER_SIZE. It is at this point that the rationale for using positive and negative values for XMTR and RCVR, respectively, can be seen most clearly. It is intuitively obvious that any transmission medium can be in one of two states, either empty or containing data. CHANNEL_BUFFER achieves these two states by either having a value of 0 or greater for BUFFER_SIZE. After the BUFFER_MODIFICATION value is applied to BUFFER_SIZE, BUFFER_SIZE is checked for a negative value. If a negative value is encountered, BUFFER_SIZE is assigned a value of 0. This operation is done to avoid having a channel which is less than empty, a concept that is not possible.

An exception is included in line 630 through 632. This is to prevent integer

overflow of BUFFER_SIZE. If such an exception occurs, BUFFER_SIZE is assigned the largest possible integer value, Max_Int.

d. Utilization Calculator

The fourth atomic operator in TRITAC_LINK is UTILIZATION_CALCULATOR. Lines 688 through 747 contains the Ada code for the implementation for UTILIZATION_CALCULATOR. Five parameters are supplied UTILIZATION_CALCULATOR. BUFFER_SIZE, DATA_PERIOD, DATA_RATE, AND START_STOP are all 'in' mode parameters. UTILIZATION is an 'out' mode parameter.

This procedure is responsible for formatting the results, as a percentage of maximum capacity, for display. A local variable, buffer_maximum, is declared and initialized by multiplying DATA_RATE AND DATA_PERIOD, as shown in line 738 of Appendix B. This yields a value which represents the maximum number of bytes the channel can accept every DATA_PERIOD at the set DATA_RATE. UTILIZATION is calculated by doing casted float division of BUFFER_SIZE by buffer_maximum, multiplied by 100, and then casted to an integer. UTILIZATION is providedas an integer value to avoid unnecessary detail being provided to the user. It is conceded that there are circumstances where this decision may give the user results which are not completely accurate, such as rounding 99.6% to 100%. Improvements to the display accuracy are topics for future work on Channel CAT.

UTILIZATION_CALCULATOR also contains statements which provide

the user text-based results. These results are displayed within the prototype's shell window.

D. OPERATION

Channel CAT can be invoked by either calling the executable image (channel_cat.exe) or by using the CAPS interface. If the CAPS interface is used, a new shell window is opened, otherwise the terminal used to execute Channel CAT will be provided all the text-based messages during execution.

1. User Input

Channel CAT starts in the paused mode. The user may select Channel CAT to run immediately, but will do so with the default value of 1 byte at a period of 1 second with a rate of 1 byte per second. More typically, the user will set DATA_SIZE, DATA_PERIOD, and DATA_RATE prior to selecting Channel CAT to run. As mentioned previously, the user is allowed to alter the parameters while Channel CAT is running.

2. Results

The value of UTILIZATION is provided not only to the results panel, but also the shell window.

a. *Text-based*

Once the user selects Channel CAT to run, the shell window will be provided a single text line on every execution of UTILIZATION_CALCULATOR. The value which is written to the shell window is the same value which is sent to

USER_INTERFACE.

b. Graphic-based

When UTILIZATION_CALCULATOR executes, UTILIZATION is sent to the results panel, specifically to the 'graph' item as shown in lines 316 through 320 of Appendix C. This value is then provided to the stripchart. The numeric value is displayed in the upper right corner of the stripchart and is also graphically plotted on the right most edge.

The resulting graphs yielded by Channel CAT are saw-toothed graphs. The peak of the graph indicates those points in time when the XMTR has completed full execution and added data to the channel. The decrease in the value displayed on the graph is caused by the incremental reduction of BUFFER_SIZE by the RCVR operator. This tracks the progress of the receipt of data and gives an indication of how much data remains to be received.

E. CHAPTER SUMMARY

The simple, clean design of Channel CAT is mostly attributable to the amount of abstraction involved. The tool is user-oriented, with several decisions being determined solely by analyzing how to provide a user with as much capability as necessary without requiring the same user to be an expert on TRI-TAC networks.

The two panel user interface, implemented in TAE+, is uncluttered and simple to use. The user is automatically provided two methods in the results panel for monitoring the modeled channel.

The TRI-TAC channel itself has been abstracted. The design concept of using an IN/OUT box to model a channel, with a transmitter adding data and a receiver removing data, was successfully implemented in the decomposed operator TRITAC_LINK.

The use of CAPS in the actual design of the Channel CAT executable was invaluable. The ability of CAPS to enforce real-time scheduling and its ability to allow a designer to rapidly create a good user interface allowed the author to complete the design phase of Channel CAT in a smooth manner.

IV. TESTING AND VALIDATION

A. TESTING

The author established the need to test and validate Channel CAT after consideration of the amount of abstraction which was involved during design. A test phase needed to be completed to illustrate the fidelity of the tool.

1. Plan

The author chose to focus solely on the accuracy of Channel CAT and not address interface issues. Two basic categories were targeted for the testing of Channel CAT. The first category focused on those situations where the parameters should yield a predictable results. The second category focused on setting the parameters at known values, establishing a control set, and then varying each parameter to measure the impact of parameter adjustment by the user.

2. Criteria

Appendix D contains the criteria the author established for the tests. The criteria is documented in the form of parameters values and the expected results. The criteria is broken into three sets of test cases. Sets 1 and 2 constitute the test cases which focus on predictable results. Set 3 contains the test cases which measure the impact of changing input parameter values by the user.

B. RESULTS

Appendix D contains the results of the tests. The author leaves the case-by-case inspection of the results to the reader. A summary of conclusions resulting from the tests

follows.

1. Accuracy

The author was primarily concerned that Channel CAT executed in a predictable manner and yielded results under known conditions which were expected. Channel CAT was found to produce predictable results. There were no variations from the expected results for all the test cases in the expected results category (Sets 1 and 2).

2. Boundary Conditions

As with any system, it is prudent to verify execution at boundary conditions. In the tests run, this involved Channel CAT being run at 100% utilization. In the three cases tested (Set 1), Channel CAT performed as expected and without deviation.

3. Fidelity

The test cases which focused on the impact of input parameter changes (Set 3) are the most interesting. The conduct of the tests required run-time manipulation of the parameters. The impact of parameter changes cannot be summarized as simply small or large, but need to be qualified for each parameter.

Changing the value of the size of data yielded different results, depending on whether a byte was added or it was subtracted from the input value. Adding a single byte to the overall size of the data caused the utilization to increase, but the change was not visible until after 5 transmission periods. This is due to the integer rounding which occurs in line 741 of Appendix B. It was determined that decimal values of .4 and below are rounded down (floor function) and decimal values of .5 or above are rounded up (ceiling

function). Subtracting a byte from the data size did not yield any change to utilization. This condition was left in place for 5 minutes with no impact seen. This is because a change of a single byte from the control conditions was not significant enough to impact the calculation of utilization. The lack of discernible change was again attributed to the integer rounding mentioned above. The author did continue to subtract a single byte until the change impacted the displayed utilization. The data size had to be decreased to 994 bytes to yield a change.

Changing the period at which data is sent over the channel by single second immediately impacted the displayed utilization. This change was present when one second was added and when it was subtracted. The change was not only immediately visible, but was significantly large. As a measure, subtracting a single second from the data period caused the utilization to grow by 11%.

The final parameter which was changed was the data rate. Adding and subtracting a single byte per second to the control conditions yielded changes in the utilization immediately. The changes, however, were not as large as the changes realized from changing the data period.

It is important to note the above conclusions are based completely on what is displayed to the user. The lack of discernible change in the utilization is not due to a software fault. As discussed in the previous chapter, the author chose to provide the results in a form which causes integer rounding and can therefor provide utilization values not fully indicative of the internal values of Channel CAT.

C. CHAPTER SUMMARY

The testing of Channel CAT was completed by the author. The testing and validation of Channel CAT focused on accuracy, boundary condition behavior, and fidelity to expected actual values. No measurements of usability of the interface were conducted.

The results of the test and validation are considered by the author to be positive. The tests for known actual results were successful and no deviations were noted. The fidelity of the test results to known actual results is important since it was the author's intent to design a tool which accurately modeled the behavior of a TRI-TAC link. The lack or delay of changes to the utilization when making small changes to the size of the data is an area which has been identified, but is also an explainable behavior.

V. CONCLUSIONS

A. STRENGTHS OF CHANNEL CAT

Channel CAT's user interface is simple and intuitive. The input panel contains explanatory labels which let the user know what parameters are available. The manipulation of the parameters is through the use of a mouse, which does not require a particular typing skill level. The presence of a button to start and pause Channel CAT is a feature which is conceptually familiar to users.

The results displayed by Channel CAT are formatted and shown on a stripchart graph. Such a graph is familiar to most people, having seen such graphs used in school classes, medical facilities, polygraphs, and seismographs. The results are also displayed numerically, which provides a more precise reading of the utilization.

The use of CAPS for designing Channel CAT provided a robust executable that performs time-critical operations predictably. The assurance of real-time behavior should be considered a major strength of Channel CAT. As the modern battlefield commander becomes more reliant on the advantages of real-time information, greater importance is placed on the development of network software systems. Through the use of an automated tool which accurately models the real-time behavior of TRI-TAC links, better design decisions can be reached earlier in the development cycle.

Finally, the development of Channel CAT within CAPS is an investment in future work and extension. The same aspects of CAPS which were found so vital in the design of Channel CAT can be leveraged for future work efforts.

B. FUTURE WORK

No software tool ever meets all users' needs and very rarely survives any length of time without needing enhanced functionality. These are recognized facts by the author and there is an expectation that Channel CAT will be the subject of future work efforts. The author has identified three significant areas for future work.

The assumptions made early in the design of Channel CAT not to include the transmission delays caused by the distance of the link should be considered for future work. As the use of satellite systems becomes more commonplace and permeates further down towards routine use by smaller military units, the delay incurred from satellite use will become a factor which must be considered. This factor could be handled by simply modifying the input parameters to include a distance parameter, but this is placing the user in the position of knowing the orbiting height of military satellites and forces them to correctly input such values. Ideally, the input panel would need to be redesigned to include a series of items which allow the user to select whether satellites are being used, how many satellites are being used, which satellites are being used, and the distance of link that is terrestrial based (not satellites).

Channel CAT currently provides utilization results in an integer format. The transition of Channel CAT to a more accurate display mode is also a strong candidate for future work. As the TRI-TAC links become more heavily used, integer rounding errors may be viewed as undesirable. The UTILIZATION_CALCULATOR would need to be modified and some modification would be necessary to the PSDL design. Such changes

are not sweeping and were not implemented in Channel CAT based on a design decision to allow initial design and implementation of Channel CAT as a rapid prototype.

The most promising and most interesting topic for future work is extension to multiple processes sharing the transmission path. Channel CAT currently provides the user the ability to model a single channel being used by two terminal devices to send and receive data. This models only a single process, which is not entirely realistic. Channel CAT needs to be extended so that the user has the ability to set a certain data rate and then provide a data size and period for several processes. This would simulate a server-to-server connection. Individual processes on servers normally do not have dedicated channels and are forced to share a transmission path. The usefulness of Channel CAT will be increased dramatically by providing functionality and interfaces for modeling several processes utilizing a single channel. The resulting graphs will certainly be more engaging than those achieved in Channel CAT's current state.

Ranking the above topics in order of importance, the inclusion of multiple processes is most important. Integrating the delay caused by distance, particularly from satellites, is the next priority. The third most important topic would be to modify the display of utilization from integer format to floating point format.

REFERENCES

[BERZ91] Berzins, V., Luqi, *Software Engineering with Abstractions*, Addison-Wesley Publishing Company, 1991.

[ELMA93] Elmasri, R., Navathe, S.B., *Fundamentals of DatabaseSystems*, 2nd ed., Benjamin/Cummings Publishing Company, Inc., 1993.

[LUQI97] Luqi, *Class Notes for CS4920*, Naval Postgraduate School, January 1997.

[MAIN91] *Maintenance Manual, AN/MRC-142*, TM-095430-30, United States Marine Corps, 1991.

[MAWT95] *Marine Air Command and Control System (MACCS) Reference Guide*, MAWTS-1 C3 Department, 1995.

BIBLIOGRAPHY

Barnes, J., *Programming in Ada 95*, Addison-Wesley, 1996.

Berzins, V., Luqi, "Semantics of a Real-Time Language," paper presented at the Real-Time Systems Symposium, Huntsville, Alabama 6-8 December 1988.

Booch, G., Bryan, D., *Software Engineering with Ada*, 3d ed, Benjamin/Cummings, Inc., 1994.

Feldman, M.B., Koffman, E.B., *Ada 95 Problem Solving and Program Design*, 2d ed, Addison-Wesley, 1996.

Hennessy, J.L., Patterson, D.A., *Computer Architecture, A Quantitative Approach*, 2d ed, Morgan Kaufmann, Inc., 1996.

Luqi, Berzins, V., Yeh, R.T., "A Prototyping Language for Real-Time Software," IEEE Transactions on Software Engineering, v. 14, no. 10, October 1988.

Luqi, Berzins, V., "Execution of a High Level Real-Time Language," paper presented at the Real-Time Systems Symposium, Huntsville, Alabama 6-8 December 1988.

Luqi, "Handling Timing Constraints in Rapid Prototyping," paper presented at the Hawaii International Conference on System Science, 22nd Kailua-Kona, Hawaii 3-6 January 1989.

Luqi, Shing, M., "Real-Time Scheduling for Software Prototyping," Journal of Systems Integration, v. 6, 1996.

Luqi, "The Role of Prototyping Languages in CASE," International Journal of Software Engineering and Knowledge Engineering, v. 1, no. 2, 1991.

Rumbaugh, J., and others, *Object-Oriented Modeling and Design*, Prentice-Hall, Inc., 1991.

Tanenbaum, A.S., *Computer Networks*, 3d ed, Prentice-Hall, Inc., 1996.

APPENDIX A. PSDL SOURCE CODE

```
OPERATOR channel_buffer
  SPECIFICATION
    INPUT
      buffer_modification : INTEGER,
      buffer_size : INTEGER,
      start_stop : BOOLEAN
    OUTPUT
      buffer_size : INTEGER
END
IMPLEMENTATION ADA channel_buffer

END

OPERATOR channel_cat
  SPECIFICATION
    STATES
      data_rate : INTEGER,
      data_period : INTEGER,
      data_size : INTEGER,
      start_stop : BOOLEAN
      INITIALLY
        1,
        1,
        1,
        FALSE
END
IMPLEMENTATION
  GRAPH
    VERTEX tritac_link

    VERTEX user_interface : 200 MS

    EDGE data_period
      user_interface ->
      tritac_link

    EDGE data_rate
      user_interface ->
      tritac_link

    EDGE data_size
      user_interface ->
      tritac_link

    EDGE start_stop
      user_interface ->
      tritac_link

    EDGE utilization
      tritac_link ->
      user_interface
  DATA STREAM
    utilization : INTEGER
  CONTROL CONSTRAINTS
    OPERATOR tritac_link
```

```
48      OPERATOR user_interface
49  END

50  OPERATOR rcvr
51    SPECIFICATION
52      INPUT
53        data_rate : INTEGER,
54        start_stop : BOOLEAN
55      OUTPUT
56        buffer_modification : INTEGER
57      MAXIMUM EXECUTION TIME 50 MS
58  END
59  IMPLEMENTATION ADA rcvr

60  END

61  OPERATOR tritac_link
62    SPECIFICATION
63      INPUT
64        data_period : INTEGER,
65        data_rate : INTEGER,
66        data_size : INTEGER,
67        start_stop : BOOLEAN
68      OUTPUT
69        utilization : INTEGER
70      STATES
71        period_counter : INTEGER,
72        buffer_size : INTEGER
73        INITIALLY
74          0,
75          0
76  END
77  IMPLEMENTATION
78    GRAPH
79      VERTEX channel_buffer

80      VERTEX rcvr : 50 MS

81      VERTEX utilization_calculator

82      VERTEX xmtr : 50 MS

83      EDGE buffer_modification
84        xmtr ->
85        channel_buffer

86      EDGE buffer_modification
87        rcvr ->
88        channel_buffer

89      EDGE buffer_size
90        channel_buffer ->
91        utilization_calculator

92      EDGE buffer_size
93        channel_buffer ->
94        channel_buffer

95      EDGE data_period
```

```
 96       EXTERNAL ->
 97       xmtr

 98     EDGE data_period
 99       EXTERNAL ->
100       utilization_calculator

101     EDGE data_rate
102       EXTERNAL ->
103       utilization_calculator

104     EDGE data_rate
105       EXTERNAL ->
106       rcvr

107     EDGE data_size
108       EXTERNAL ->
109       xmtr

110     EDGE period_counter
111       xmtr ->
112       xmtr

113     EDGE start_stop
114       EXTERNAL ->
115       channel_buffer

116     EDGE start_stop
117       EXTERNAL ->
118       rcvr

119     EDGE start_stop
120       EXTERNAL ->
121       xmtr

122     EDGE start_stop
123       EXTERNAL ->
124       utilization_calculator

125     EDGE utilization
126       utilization_calculator ->
127       EXTERNAL
128   DATA STREAM
129     buffer_modification : INTEGER
130   CONTROL CONSTRAINTS
131     OPERATOR channel_buffer
132       TRIGGERED BY SOME
133         buffer_modification
134       PERIOD 500 MS

135     OPERATOR rcvr
136       TRIGGERED IF
137         start_stop
138       PERIOD 500 MS

139     OPERATOR utilization_calculator
140       TRIGGERED BY SOME
141         buffer_size
```

```
142     OPERATOR xmtr
143       TRIGGERED IF
144         start_stop
145       PERIOD 1000 MS
146 END

147 OPERATOR user_interface
148   SPECIFICATION
149     INPUT
150       utilization : INTEGER
151     OUTPUT
152       data_period : INTEGER,
153       data_rate : INTEGER,
154       data_size : INTEGER,
155       start_stop : BOOLEAN
156     MAXIMUM EXECUTION TIME 200 MS
157 END
158 IMPLEMENTATION ADA user_interface

159 END

160 OPERATOR utilization_calculator
161   SPECIFICATION
162     INPUT
163       buffer_size : INTEGER,
164       data_period : INTEGER,
165       data_rate : INTEGER,
166       start_stop : BOOLEAN
167     OUTPUT
168       utilization : INTEGER
169 END
170 IMPLEMENTATION ADA utilization_calculator

171 END

172 OPERATOR xmtr
173   SPECIFICATION
174     INPUT
175       data_period : INTEGER,
176       data_size : INTEGER,
177       period_counter : INTEGER,
178       start_stop : BOOLEAN
179     OUTPUT
180       buffer_modification : INTEGER,
181       period_counter : INTEGER
182     MAXIMUM EXECUTION TIME 50 MS
183 END
184 IMPLEMENTATION ADA xmtr

185 END
```

APPENDIX B. ADA SOURCE CODE

```
1   package CHANNEL_CAT_EXCEPTIONS is
2     -- PSDL exception type declaration
3     type PSDL_EXCEPTION is (UNDECLARED_ADA_EXCEPTION);
4   end CHANNEL_CAT_EXCEPTIONS;

5   package CHANNEL_CAT_INSTANTIATIONS is
6     -- Ada Generic package instantiations

7   end CHANNEL_CAT_INSTANTIATIONS;

8     with PSDL_TIMERS;
9   package CHANNEL_CAT_TIMERS is
10    -- Timer instantiations
11  end CHANNEL_CAT_TIMERS;

12 -- with/use clauses for atomic type packages
13 -- with/use clauses for generated packages.
14    with CHANNEL_CAT_EXCEPTIONS; use CHANNEL_CAT_EXCEPTIONS;
15    with CHANNEL_CAT_INSTANTIATIONS; use CHANNEL_CAT_INSTANTIATIONS;
16 -- with/use clauses for CAPS library packages.
17    with PSDL_STREAMS; use PSDL_STREAMS;
18  package CHANNEL_CAT_STREAMS is
19 -- Local stream instantiations

20    package DS_UTILIZATION_USER_INTERFACE is new
21      PSDL_STREAMS.SAMPLED_BUFFER(INTEGER);

22    package DS_BUFFER_MODIFICATION_CHANNEL_BUFFER is new
23      PSDL_STREAMS.SAMPLED_BUFFER(INTEGER);

24 -- State stream instantiations

25    package DS_DATA_RATE_UTILIZATION_CALCULATOR is new
26      PSDL_STREAMS.STATE_VARIABLE(INTEGER, 1);

27    package DS_DATA_RATE_RCVR is new
28      PSDL_STREAMS.STATE_VARIABLE(INTEGER, 1);

29    package DS_DATA_PERIOD_XMTR is new
30      PSDL_STREAMS.STATE_VARIABLE(INTEGER, 1);

31    package DS_DATA_PERIOD_UTILIZATION_CALCULATOR is new
32      PSDL_STREAMS.STATE_VARIABLE(INTEGER, 1);

33    package DS_DATA_SIZE_XMTR is new
34      PSDL_STREAMS.STATE_VARIABLE(INTEGER, 1);

35    package DS_START_STOP_XMTR is new
36      PSDL_STREAMS.STATE_VARIABLE(BOOLEAN, false);

37    package DS_START_STOP_UTILIZATION_CALCULATOR is new
38      PSDL_STREAMS.STATE_VARIABLE(BOOLEAN, false);

39    package DS_START_STOP_RCVR is new
40      PSDL_STREAMS.STATE_VARIABLE(BOOLEAN, false);
```

```
41      package DS_START_STOP_CHANNEL_BUFFER is new
42        PSDL_STREAMS.STATE_VARIABLE(BOOLEAN, false);

43      package DS_PERIOD_COUNTER_XMTR is new
44        PSDL_STREAMS.STATE_VARIABLE(INTEGER, 0);

45      package DS_BUFFER_SIZE_CHANNEL_BUFFER is new
46        PSDL_STREAMS.STATE_VARIABLE(INTEGER, 0);

47      package DS_BUFFER_SIZE_UTILIZATION_CALCULATOR is new
48        PSDL_STREAMS.STATE_VARIABLE(INTEGER, 0);

49    end CHANNEL_CAT_STREAMS;

50    package CHANNEL_CAT_DRIVERS is
51      procedure USER_INTERFACE_DRIVER;
52      procedure CHANNEL_BUFFER_DRIVER;
53      procedure RCVR_DRIVER;
54      procedure UTILIZATION_CALCULATOR_DRIVER;
55      procedure XMTR_DRIVER;
56    end CHANNEL_CAT_DRIVERS;

57  -- with/use clauses for atomic components.
58      with CHANNEL_BUFFER_PKG; use CHANNEL_BUFFER_PKG;
59      with RCVR_PKG; use RCVR_PKG;
60      with USER_INTERFACE_PKG; use USER_INTERFACE_PKG;
61      with UTILIZATION_CALCULATOR_PKG; use UTILIZATION_CALCULATOR_PKG;
62      with XMTR_PKG; use XMTR_PKG;
63  -- with/use clauses for generated packages.
64      with CHANNEL_CAT_EXCEPTIONS; use CHANNEL_CAT_EXCEPTIONS;
65      with CHANNEL_CAT_STREAMS; use CHANNEL_CAT_STREAMS;
66      with CHANNEL_CAT_TIMERS; use CHANNEL_CAT_TIMERS;
67      with CHANNEL_CAT_INSTANTIATIONS; use CHANNEL_CAT_INSTANTIATIONS;
68  -- with/use clauses for CAPS library packages.
69      with DS_DEBUG_PKG; use DS_DEBUG_PKG;
70      with PSDL_STREAMS; use PSDL_STREAMS;
71      with PSDL_TIMERS;
72    package body CHANNEL_CAT_DRIVERS is

73      procedure USER_INTERFACE_DRIVER is
74        LV_UTILIZATION : INTEGER;
75        LV_DATA_PERIOD : INTEGER;
76        LV_DATA_RATE : INTEGER;
77        LV_DATA_SIZE : INTEGER;
78        LV_START_STOP : BOOLEAN;

79        EXCEPTION_HAS_OCCURRED: BOOLEAN := FALSE;
80        EXCEPTION_ID: PSDL_EXCEPTION;
81      begin
82  -- Data trigger checks.

83  -- Data stream reads.
84        begin
85          DS_UTILIZATION_USER_INTERFACE.BUFFER.READ(LV_UTILIZATION);
86        exception
87          when BUFFER_UNDERFLOW =>
88            DS_DEBUG.BUFFER_UNDERFLOW("UTILIZATION_USER_INTERFACE", "USER_INTERFACE");
89        end;
```

```
90 -- Execution trigger condition check.
91       if True then
92         begin
93         USER_INTERFACE(
94           UTILIZATION => LV_UTILIZATION,
95           DATA_PERIOD => LV_DATA_PERIOD,
96           DATA_RATE => LV_DATA_RATE,
97           DATA_SIZE => LV_DATA_SIZE,
98           START_STOP => LV_START_STOP);
99         exception
100          when others =>
101            DS_DEBUG.UNDECLARED_EXCEPTION("USER_INTERFACE");
102            EXCEPTION_HAS_OCCURRED := true;
103            EXCEPTION_ID := UNDECLARED_ADA_EXCEPTION;
104        end;
105      else return;
106      end if;

107  -- Exception Constraint translations.

108  -- Other constraint option translations.

109  --Unconditional output translations.
110      if not EXCEPTION_HAS_OCCURRED then
111        begin
112          DS_DATA_PERIOD_XMTR.BUFFER.WRITE(LV_DATA_PERIOD);
113        exception
114          when BUFFER_OVERFLOW =>
115            DS_DEBUG.BUFFER_OVERFLOW("DATA_PERIOD_XMTR", "USER_INTERFACE");
116        end;
117        begin
118          DS_DATA_PERIOD_UTILIZATION_CALCULATOR.BUFFER.WRITE(LV_DATA_PERIOD);
119        exception
120          when BUFFER_OVERFLOW =>
121            DS_DEBUG.BUFFER_OVERFLOW("DATA_PERIOD_UTILIZATION_CALCULATOR",
122 "USER_INTERFACE");
123        end;
124      end if;
125      if not EXCEPTION_HAS_OCCURRED then
126        begin
127          DS_DATA_RATE_UTILIZATION_CALCULATOR.BUFFER.WRITE(LV_DATA_RATE);
128        exception
129          when BUFFER_OVERFLOW =>
130            DS_DEBUG.BUFFER_OVERFLOW("DATA_RATE_UTILIZATION_CALCULATOR",
131 "USER_INTERFACE");
132        end;
133        begin
134          DS_DATA_RATE_RCVR.BUFFER.WRITE(LV_DATA_RATE);
135        exception
136          when BUFFER_OVERFLOW =>
137            DS_DEBUG.BUFFER_OVERFLOW("DATA_RATE_RCVR", "USER_INTERFACE");
138        end;
139      end if;
140      if not EXCEPTION_HAS_OCCURRED then
141        begin
142          DS_DATA_SIZE_XMTR.BUFFER.WRITE(LV_DATA_SIZE);
143        exception
144          when BUFFER_OVERFLOW =>
145            DS_DEBUG.BUFFER_OVERFLOW("DATA_SIZE_XMTR", "USER_INTERFACE");
```

```
146         end;
147       end if;
148       if not EXCEPTION_HAS_OCCURRED then
149         begin
150           DS_START_STOP_XMTR.BUFFER.WRITE(LV_START_STOP);
151         exception
152           when BUFFER_OVERFLOW =>
153             DS_DEBUG.BUFFER_OVERFLOW("START_STOP_XMTR", "USER_INTERFACE");
154         end;
155         begin
156           DS_START_STOP_UTILIZATION_CALCULATOR.BUFFER.WRITE(LV_START_STOP);
157         exception
158           when BUFFER_OVERFLOW =>
159             DS_DEBUG.BUFFER_OVERFLOW("START_STOP_UTILIZATION_CALCULATOR",
160 "USER_INTERFACE");
161         end;
162         begin
163           DS_START_STOP_RCVR.BUFFER.WRITE(LV_START_STOP);
164         exception
165           when BUFFER_OVERFLOW =>
166             DS_DEBUG.BUFFER_OVERFLOW("START_STOP_RCVR", "USER_INTERFACE");
167         end;
168         begin
169           DS_START_STOP_CHANNEL_BUFFER.BUFFER.WRITE(LV_START_STOP);
170         exception
171           when BUFFER_OVERFLOW =>
172             DS_DEBUG.BUFFER_OVERFLOW("START_STOP_CHANNEL_BUFFER", "USER_INTERFACE");
173         end;
174       end if;

175  -- PSDL Exception handler.
176       if EXCEPTION_HAS_OCCURRED then
177         DS_DEBUG.UNHANDLED_EXCEPTION(
178           "USER_INTERFACE",
179           PSDL_EXCEPTION'IMAGE(EXCEPTION_ID));
180       end if;
181     end USER_INTERFACE_DRIVER;


182     procedure CHANNEL_BUFFER_DRIVER is
183       LV_BUFFER_MODIFICATION : INTEGER;
184       LV_START_STOP : BOOLEAN;
185       LV_BUFFER_SIZE : INTEGER;

186       EXCEPTION_HAS_OCCURRED: BOOLEAN := FALSE;
187       EXCEPTION_ID: PSDL_EXCEPTION;
188     begin
189  -- Data trigger checks.
190       if not (DS_BUFFER_MODIFICATION_CHANNEL_BUFFER.NEW_DATA) then
191         return;
192       end if;

193  -- Data stream reads.
194       begin
195         DS_BUFFER_MODIFICATION_CHANNEL_BUFFER.BUFFER.READ(LV_BUFFER_MODIFICATION);
196       exception
197         when BUFFER_UNDERFLOW =>
198           DS_DEBUG.BUFFER_UNDERFLOW("BUFFER_MODIFICATION_CHANNEL_BUFFER",
199 "CHANNEL_BUFFER");
```

```
200       end;
201       begin
202         DS_BUFFER_SIZE_CHANNEL_BUFFER.BUFFER.READ(LV_BUFFER_SIZE);
203       exception
204         when BUFFER_UNDERFLOW =>
205           DS_DEBUG.BUFFER_UNDERFLOW("BUFFER_SIZE_CHANNEL_BUFFER", "CHANNEL_BUFFER");
206       end;
207       begin
208         DS_START_STOP_CHANNEL_BUFFER.BUFFER.READ(LV_START_STOP);
209       exception
210         when BUFFER_UNDERFLOW =>
211           DS_DEBUG.BUFFER_UNDERFLOW("START_STOP_CHANNEL_BUFFER", "CHANNEL_BUFFER");
212       end;

213 -- Execution trigger condition check.
214       if True then
215         begin
216         CHANNEL_BUFFER(
217           BUFFER_MODIFICATION => LV_BUFFER_MODIFICATION,
218           START_STOP => LV_START_STOP,
219           BUFFER_SIZE => LV_BUFFER_SIZE);
220         exception
221           when others =>
222             DS_DEBUG.UNDECLARED_EXCEPTION("CHANNEL_BUFFER");
223             EXCEPTION_HAS_OCCURRED := true;
224             EXCEPTION_ID := UNDECLARED_ADA_EXCEPTION;
225         end;
226       else return;
227       end if;

228  -- Exception Constraint translations.

229  -- Other constraint option translations.

230  --Unconditional output translations.
231       if not EXCEPTION_HAS_OCCURRED then
232         begin
233           DS_BUFFER_SIZE_CHANNEL_BUFFER.BUFFER.WRITE(LV_BUFFER_SIZE);
234         exception
235           when BUFFER_OVERFLOW =>
236             DS_DEBUG.BUFFER_OVERFLOW("BUFFER_SIZE_CHANNEL_BUFFER", "CHANNEL_BUFFER");
237         end;
238         begin
239           DS_BUFFER_SIZE_UTILIZATION_CALCULATOR.BUFFER.WRITE(LV_BUFFER_SIZE);
240         exception
241           when BUFFER_OVERFLOW =>
242             DS_DEBUG.BUFFER_OVERFLOW("BUFFER_SIZE_UTILIZATION_CALCULATOR",
243 "CHANNEL_BUFFER");
244         end;
245       end if;

246  -- PSDL Exception handler.
247       if EXCEPTION_HAS_OCCURRED then
248         DS_DEBUG.UNHANDLED_EXCEPTION(
249           "CHANNEL_BUFFER",
250           PSDL_EXCEPTION'IMAGE(EXCEPTION_ID));
251       end if;
252     end CHANNEL_BUFFER_DRIVER;
```

```
253     procedure RCVR_DRIVER is
254       LV_DATA_RATE : INTEGER;
255       LV_START_STOP : BOOLEAN;
256       LV_BUFFER_MODIFICATION : INTEGER;

257       EXCEPTION_HAS_OCCURRED: BOOLEAN := FALSE;
258       EXCEPTION_ID: PSDL_EXCEPTION;
259     begin
260 -- Data trigger checks.

261 -- Data stream reads.
262       begin
263         DS_DATA_RATE_RCVR.BUFFER.READ(LV_DATA_RATE);
264       exception
265         when BUFFER_UNDERFLOW =>
266           DS_DEBUG.BUFFER_UNDERFLOW("DATA_RATE_RCVR", "RCVR");
267       end;
268       begin
269         DS_START_STOP_RCVR.BUFFER.READ(LV_START_STOP);
270       exception
271         when BUFFER_UNDERFLOW =>
272           DS_DEBUG.BUFFER_UNDERFLOW("START_STOP_RCVR", "RCVR");
273       end;

274 -- Execution trigger condition check.
275       if LV_START_STOP then
276         begin
277         RCVR(
278           DATA_RATE => LV_DATA_RATE,
279           START_STOP => LV_START_STOP,
280           BUFFER_MODIFICATION => LV_BUFFER_MODIFICATION);
281         exception
282           when others =>
283             DS_DEBUG.UNDECLARED_EXCEPTION("RCVR");
284             EXCEPTION_HAS_OCCURRED := true;
285             EXCEPTION_ID := UNDECLARED_ADA_EXCEPTION;
286         end;
287       else return;
288       end if;

289  -- Exception Constraint translations.

290  -- Other constraint option translations.

291  --Unconditional output translations.
292       if not EXCEPTION_HAS_OCCURRED then
293         begin
294           DS_BUFFER_MODIFICATION_CHANNEL_BUFFER.BUFFER.WRITE(LV_BUFFER_MODIFICATION);
295         exception
296           when BUFFER_OVERFLOW =>
297             DS_DEBUG.BUFFER_OVERFLOW("BUFFER_MODIFICATION_CHANNEL_BUFFER", "RCVR");
298         end;
299       end if;

300  -- PSDL Exception handler.
301       if EXCEPTION_HAS_OCCURRED then
302         DS_DEBUG.UNHANDLED_EXCEPTION(
303           "RCVR",
```

```
304           PSDL_EXCEPTION'IMAGE(EXCEPTION_ID));
305       end if;
306     end RCVR_DRIVER;


307     procedure UTILIZATION_CALCULATOR_DRIVER is
308       LV_BUFFER_SIZE : INTEGER;
309       LV_DATA_PERIOD : INTEGER;
310       LV_DATA_RATE : INTEGER;
311       LV_START_STOP : BOOLEAN;
312       LV_UTILIZATION : INTEGER;

313       EXCEPTION_HAS_OCCURRED: BOOLEAN := FALSE;
314       EXCEPTION_ID: PSDL_EXCEPTION;
315     begin
316 -- Data trigger checks.
317       if not (DS_BUFFER_SIZE_UTILIZATION_CALCULATOR.NEW_DATA) then
318         return;
319       end if;

320 -- Data stream reads.
321       begin
322         DS_BUFFER_SIZE_UTILIZATION_CALCULATOR.BUFFER.READ(LV_BUFFER_SIZE);
323       exception
324         when BUFFER_UNDERFLOW =>
325           DS_DEBUG.BUFFER_UNDERFLOW("BUFFER_SIZE_UTILIZATION_CALCULATOR",
326 "UTILIZATION_CALCULATOR");
327       end;
328       begin
329         DS_DATA_PERIOD_UTILIZATION_CALCULATOR.BUFFER.READ(LV_DATA_PERIOD);
330       exception
331         when BUFFER_UNDERFLOW =>
332           DS_DEBUG.BUFFER_UNDERFLOW("DATA_PERIOD_UTILIZATION_CALCULATOR",
333 "UTILIZATION_CALCULATOR");
334       end;
335       begin
336         DS_DATA_RATE_UTILIZATION_CALCULATOR.BUFFER.READ(LV_DATA_RATE);
337       exception
338         when BUFFER_UNDERFLOW =>
339           DS_DEBUG.BUFFER_UNDERFLOW("DATA_RATE_UTILIZATION_CALCULATOR",
340 "UTILIZATION_CALCULATOR");
341       end;
342       begin
343         DS_START_STOP_UTILIZATION_CALCULATOR.BUFFER.READ(LV_START_STOP);
344       exception
345         when BUFFER_UNDERFLOW =>
346           DS_DEBUG.BUFFER_UNDERFLOW("START_STOP_UTILIZATION_CALCULATOR",
347 "UTILIZATION_CALCULATOR");
348       end;

349 -- Execution trigger condition check.
350       if True then
351         begin
352         UTILIZATION_CALCULATOR(
353           BUFFER_SIZE => LV_BUFFER_SIZE,
354           DATA_PERIOD => LV_DATA_PERIOD,
355           DATA_RATE => LV_DATA_RATE,
356           START_STOP => LV_START_STOP,
357           UTILIZATION => LV_UTILIZATION);
```

```
358         exception
359           when others =>
360             DS_DEBUG.UNDECLARED_EXCEPTION("UTILIZATION_CALCULATOR");
361             EXCEPTION_HAS_OCCURRED := true;
362             EXCEPTION_ID := UNDECLARED_ADA_EXCEPTION;
363         end;
364       else return;
365       end if;

366  -- Exception Constraint translations.

367  -- Other constraint option translations.

368  --Unconditional output translations.
369       if not EXCEPTION_HAS_OCCURRED then
370         begin
371           DS_UTILIZATION_USER_INTERFACE.BUFFER.WRITE(LV_UTILIZATION);
372         exception
373           when BUFFER_OVERFLOW =>
374             DS_DEBUG.BUFFER_OVERFLOW("UTILIZATION_USER_INTERFACE",
375 "UTILIZATION_CALCULATOR");
376         end;
377       end if;

378  -- PSDL Exception handler.
379       if EXCEPTION_HAS_OCCURRED then
380         DS_DEBUG.UNHANDLED_EXCEPTION(
381           "UTILIZATION_CALCULATOR",
382           PSDL_EXCEPTION'IMAGE(EXCEPTION_ID));
383       end if;
384     end UTILIZATION_CALCULATOR_DRIVER;


385     procedure XMTR_DRIVER is
386       LV_DATA_PERIOD : INTEGER;
387       LV_DATA_SIZE : INTEGER;
388       LV_START_STOP : BOOLEAN;
389       LV_BUFFER_MODIFICATION : INTEGER;
390       LV_PERIOD_COUNTER : INTEGER;

391       EXCEPTION_HAS_OCCURRED: BOOLEAN := FALSE;
392       EXCEPTION_ID: PSDL_EXCEPTION;
393     begin
394  -- Data trigger checks.

395  -- Data stream reads.
396       begin
397         DS_DATA_PERIOD_XMTR.BUFFER.READ(LV_DATA_PERIOD);
398       exception
399         when BUFFER_UNDERFLOW =>
400           DS_DEBUG.BUFFER_UNDERFLOW("DATA_PERIOD_XMTR", "XMTR");
401       end;
402       begin
403         DS_DATA_SIZE_XMTR.BUFFER.READ(LV_DATA_SIZE);
404       exception
405         when BUFFER_UNDERFLOW =>
406           DS_DEBUG.BUFFER_UNDERFLOW("DATA_SIZE_XMTR", "XMTR");
407       end;
408       begin
```

```
409         DS_PERIOD_COUNTER_XMTR.BUFFER.READ(LV_PERIOD_COUNTER);
410       exception
411         when BUFFER_UNDERFLOW =>
412           DS_DEBUG.BUFFER_UNDERFLOW("PERIOD_COUNTER_XMTR", "XMTR");
413       end;
414       begin
415         DS_START_STOP_XMTR.BUFFER.READ(LV_START_STOP);
416       exception
417         when BUFFER_UNDERFLOW =>
418           DS_DEBUG.BUFFER_UNDERFLOW("START_STOP_XMTR", "XMTR");
419       end;

420 -- Execution trigger condition check.
421       if LV_START_STOP then
422         begin
423         XMTR(
424           DATA_PERIOD => LV_DATA_PERIOD,
425           DATA_SIZE => LV_DATA_SIZE,
426           START_STOP => LV_START_STOP,
427           BUFFER_MODIFICATION => LV_BUFFER_MODIFICATION,
428           PERIOD_COUNTER => LV_PERIOD_COUNTER);
429         exception
430           when others =>
431             DS_DEBUG.UNDECLARED_EXCEPTION("XMTR");
432             EXCEPTION_HAS_OCCURRED := true;
433             EXCEPTION_ID := UNDECLARED_ADA_EXCEPTION;
434         end;
435       else return;
436       end if;

437 -- Exception Constraint translations.

438 -- Other constraint option translations.

439 --Unconditional output translations.
440       if not EXCEPTION_HAS_OCCURRED then
441         begin
442           DS_BUFFER_MODIFICATION_CHANNEL_BUFFER.BUFFER.WRITE(LV_BUFFER_MODIFICATION);
443         exception
444           when BUFFER_OVERFLOW =>
445             DS_DEBUG.BUFFER_OVERFLOW("BUFFER_MODIFICATION_CHANNEL_BUFFER", "XMTR");
446         end;
447       end if;
448       if not EXCEPTION_HAS_OCCURRED then
449         begin
450           DS_PERIOD_COUNTER_XMTR.BUFFER.WRITE(LV_PERIOD_COUNTER);
451         exception
452           when BUFFER_OVERFLOW =>
453             DS_DEBUG.BUFFER_OVERFLOW("PERIOD_COUNTER_XMTR", "XMTR");
454         end;
455       end if;

456 -- PSDL Exception handler.
457       if EXCEPTION_HAS_OCCURRED then
458         DS_DEBUG.UNHANDLED_EXCEPTION(
459           "XMTR",
460           PSDL_EXCEPTION'IMAGE(EXCEPTION_ID));
461       end if;
462     end XMTR_DRIVER;
```

```
463   end CHANNEL_CAT_DRIVERS;

464   package channel_cat_DYNAMIC_SCHEDULERS is
465     procedure START_DYNAMIC_SCHEDULE;
466   end channel_cat_DYNAMIC_SCHEDULERS;


467   with channel_cat_DRIVERS;  use channel_cat_DRIVERS;
468   with PRIORITY_DEFINITIONS;  use PRIORITY_DEFINITIONS;
469   package body channel_cat_DYNAMIC_SCHEDULERS is

470     task type DYNAMIC_SCHEDULE_TYPE is
471       pragma priority (DYNAMIC_SCHEDULE_PRIORITY);
472       entry START;
473     end DYNAMIC_SCHEDULE_TYPE;
474     for DYNAMIC_SCHEDULE_TYPE'STORAGE_SIZE use 100_000;
475     DYNAMIC_SCHEDULE : DYNAMIC_SCHEDULE_TYPE;

476     task body DYNAMIC_SCHEDULE_TYPE is
477     begin
478       accept START;
479       loop
480         utilization_calculator_DRIVER;
481         channel_buffer_DRIVER;
482       end loop;
483     end DYNAMIC_SCHEDULE_TYPE;

484     procedure START_DYNAMIC_SCHEDULE is
485     begin
486       DYNAMIC_SCHEDULE.START;
487     end START_DYNAMIC_SCHEDULE;

488   end channel_cat_DYNAMIC_SCHEDULERS;


489   package channel_cat_STATIC_SCHEDULERS is
490     procedure START_STATIC_SCHEDULE;
491   end channel_cat_STATIC_SCHEDULERS;

492   with channel_cat_DRIVERS;  use channel_cat_DRIVERS;
493   with PRIORITY_DEFINITIONS;  use PRIORITY_DEFINITIONS;
494   with PSDL_TIMERS;  use PSDL_TIMERS;
495   with TEXT_IO;  use TEXT_IO;
496   package body channel_cat_STATIC_SCHEDULERS is

497     task type STATIC_SCHEDULE_TYPE is
498       pragma priority (STATIC_SCHEDULE_PRIORITY);
499       entry START;
500     end STATIC_SCHEDULE_TYPE;
501     for STATIC_SCHEDULE_TYPE'STORAGE_SIZE use 200_000;
502     STATIC_SCHEDULE : STATIC_SCHEDULE_TYPE;

503     task body STATIC_SCHEDULE_TYPE is
504       PERIOD : duration;
505       user_interface_START_TIME1 : duration;
506       user_interface_STOP_TIME1 : duration;
507       rcvr_START_TIME2 : duration;
508       rcvr_STOP_TIME2 : duration;
509       xmtr_START_TIME3 : duration;
```

```
510       xmtr_STOP_TIME3 : duration;
511       user_interface_START_TIME4 : duration;
512       user_interface_STOP_TIME4 : duration;
513       rcvr_START_TIME5 : duration;
514       rcvr_STOP_TIME5 : duration;
515       schedule_timer : TIMER := NEW_TIMER;
516     begin
517       accept START;
518       PERIOD := TARGET_TO_HOST(duration( 1.00000000000000E+00));
519       user_interface_START_TIME1 := TARGET_TO_HOST(duration( 0.00000000000000E+00));
520       user_interface_STOP_TIME1 := TARGET_TO_HOST(duration( 2.00000000000000E-01));
521       rcvr_START_TIME2 := TARGET_TO_HOST(duration( 2.00000000000000E-01));
522       rcvr_STOP_TIME2 := TARGET_TO_HOST(duration( 2.50000000000000E-01));
523       xmtr_START_TIME3 := TARGET_TO_HOST(duration( 2.50000000000000E-01));
524       xmtr_STOP_TIME3 := TARGET_TO_HOST(duration( 3.00000000000000E-01));
525       user_interface_START_TIME4 := TARGET_TO_HOST(duration( 5.00000000000000E-01));
526       user_interface_STOP_TIME4 := TARGET_TO_HOST(duration( 7.00000000000000E-01));
527       rcvr_START_TIME5 := TARGET_TO_HOST(duration( 7.00000000000000E-01));
528       rcvr_STOP_TIME5 := TARGET_TO_HOST(duration( 7.50000000000000E-01));
529       START(schedule_timer);
530       loop
531         delay(user_interface_START_TIME1 - HOST_DURATION(schedule_timer));
532         user_interface_DRIVER;
533         if HOST_DURATION(schedule_timer) > user_interface_STOP_TIME1 then
534           PUT_LINE("timing error from operator user_interface");
535           SUBTRACT_HOST_TIME_FROM_ALL_TIMERS(HOST_DURATION(schedule_timer) -
536 user_interface_STOP_TIME1);
537         end if;

538         delay(rcvr_START_TIME2 - HOST_DURATION(schedule_timer));
539         rcvr_DRIVER;
540         if HOST_DURATION(schedule_timer) > rcvr_STOP_TIME2 then
541           PUT_LINE("timing error from operator rcvr");
542           SUBTRACT_HOST_TIME_FROM_ALL_TIMERS(HOST_DURATION(schedule_timer) -
543 rcvr_STOP_TIME2);
544         end if;

545         delay(xmtr_START_TIME3 - HOST_DURATION(schedule_timer));
546         xmtr_DRIVER;
547         if HOST_DURATION(schedule_timer) > xmtr_STOP_TIME3 then
548           PUT_LINE("timing error from operator xmtr");
549           SUBTRACT_HOST_TIME_FROM_ALL_TIMERS(HOST_DURATION(schedule_timer) -
550 xmtr_STOP_TIME3);
551         end if;

552         delay(user_interface_START_TIME4 - HOST_DURATION(schedule_timer));
553         user_interface_DRIVER;
554         if HOST_DURATION(schedule_timer) > user_interface_STOP_TIME4 then
555           PUT_LINE("timing error from operator user_interface");
556           SUBTRACT_HOST_TIME_FROM_ALL_TIMERS(HOST_DURATION(schedule_timer) -
557 user_interface_STOP_TIME4);
558         end if;

559         delay(rcvr_START_TIME5 - HOST_DURATION(schedule_timer));
560         rcvr_DRIVER;
561         if HOST_DURATION(schedule_timer) > rcvr_STOP_TIME5 then
562           PUT_LINE("timing error from operator rcvr");
563           SUBTRACT_HOST_TIME_FROM_ALL_TIMERS(HOST_DURATION(schedule_timer) -
564 rcvr_STOP_TIME5);
```

```
565         end if;

566         delay(PERIOD - HOST_DURATION(schedule_timer));
567         RESET(schedule_timer);
568       end loop;
569     end STATIC_SCHEDULE_TYPE;

570     procedure START_STATIC_SCHEDULE is
571     begin
572       STATIC_SCHEDULE.START;
573     end START_STATIC_SCHEDULE;

574   end channel_cat_STATIC_SCHEDULERS;


575     with CHANNEL_CAT_STATIC_SCHEDULERS; use CHANNEL_CAT_STATIC_SCHEDULERS;
576     with CHANNEL_CAT_DYNAMIC_SCHEDULERS; use CHANNEL_CAT_DYNAMIC_SCHEDULERS;
577     with CAPS_HARDWARE_MODEL; use CAPS_HARDWARE_MODEL;
578   procedure CHANNEL_CAT is
579   begin
580     init_hardware_model;
581     start_static_schedule;
582     start_dynamic_schedule;
583   end CHANNEL_CAT;


584 ------------------------------------------------------------
585 --File : CHANNEL_BUFFER_PKG
586 --Author : Mike Coleman, CAPT, USMC
587 --Project : Channel CAT (Thesis project)
588 --Date : September 1997
589 --Notes : Any personal comment by author will use
590 --        standard comments followed by initials (mgc).
591 --Description : This package is designed to implement the
592 --              simulation of a TRI-TAC digital channel by
593 --              using an integer buffer.  This buffer is
594 --              subject to modification by two external
595 --              operators, xmtr and rcvr.  These two
596 --              send identical streams to the buffer, with
597 --              xmtr providing a positive integer and rcvr
598 --              providing a negative integer value.  The
599 --              value of the buffer is output upon
600 --              modification.
601 ------------------------------------------------------------
602 with System;use System;
603 with Text_IO;

604 -- Specification for CHANNEL_BUFFER_PKG

605 package CHANNEL_BUFFER_PKG is


606     -- Procedure: CHANNEL_BUFFER
607     -- Purpose : Operator which acts as the "channel" in Channel CAT.
608     -- Parameters: BUFFER_MODIFICATION : an integer data type which
609     --                                   can be received from either the
610     --                                   rcvr or xmtr.
611     --             START_STOP :  a boolean value used to designate
612     --                           whether the channel should be running or
```

```
613    --                          paused (paused refers to being "froze").
614    procedure CHANNEL_BUFFER
615       (BUFFER_MODIFICATION : IN INTEGER;
616        START_STOP : IN BOOLEAN;
617        BUFFER_SIZE : IN OUT INTEGER);

618 end CHANNEL_BUFFER_PKG;


619 -- Body for CHANNEL_BUFFER_PKG

620 package body CHANNEL_BUFFER_PKG is

621    procedure CHANNEL_BUFFER
622       (BUFFER_MODIFICATION : IN INTEGER;
623        START_STOP : IN BOOLEAN;
624        BUFFER_SIZE : IN OUT INTEGER) is
625    begin
626       BUFFER_SIZE := BUFFER_SIZE + BUFFER_MODIFICATION;
627       if (BUFFER_SIZE < 0) then
628          BUFFER_SIZE := 0;
629       end if;
630    exception
631       when Numeric_Error =>
632          BUFFER_SIZE := Max_Int;
633    end CHANNEL_BUFFER;
634 end CHANNEL_BUFFER_PKG;


635 ------------------------------------------------------------
636 --File : RCVR_PKG
637 --Author : Mike Coleman, CAPT, USMC
638 --Project : Channel CAT (Thesis project)
639 --Date : September 1997
640 --Notes : Any personal comment by author will use
641 --        standard comments followed by initials (mgc).
642 --Description : This package is designed to simulate the
643 --              portion of a TRI-TAC responsible for the
644 --              receipt of data on the channel (receiver).
645 --              This is done by sending channel a negative
646 --              number to be applied to the buffer, thereby
647 --              simulating receipt of data and it removal
648 --              from the channel.  The amount which is
649 --              "removed" is dependent upon the data_rate,
650 --              which supplied by the user.
651 ------------------------------------------------------------
652 with System;use System;
653 with Text_IO;

654 -- Specification for RCVR_PKG

655 package rcvr_pkg is

656     -- Procedure: RCVR
657     -- Purpose : Operator which acts as the "receiver" in Channel CAT.
658     -- Parameters: DATA_RATE : an integer value which represents the
659     --                         total size in bytes of the data element
660     --                         which being transmitted.
661     --             START_STOP :  a boolean value used to designate
```

```
662     --                              whether the channel should be running or
663     --                              paused (paused refers to being "froze").
664     --              BUFFER_MODIFICATION : an integer data type which
665     --                                    is sent to CHANNEL_BUFFER.
666    procedure RCVR(DATA_RATE : IN INTEGER;
667                   START_STOP : IN BOOLEAN;
668                   BUFFER_MODIFICATION : OUT INTEGER);

669 end rcvr_pkg;



670 -- Body for RCVR_PKG

671 package body rcvr_pkg is

672    package Util_IO is new Text_IO.Integer_IO(Integer);

673    procedure RCVR (DATA_RATE : IN INTEGER;
674                    START_STOP : IN BOOLEAN;
675                    BUFFER_MODIFICATION : OUT INTEGER) is
676    ops_per_second : integer := 2;
677    amount_per_500_ms : integer;
678    BEGIN
679       if DATA_RATE > 2 then
680          amount_per_500_ms := (DATA_RATE) /
681                          ops_per_second;
682          BUFFER_MODIFICATION := -amount_per_500_ms;
683       else
684          BUFFER_MODIFICATION := -1;
685       end if;
686    end RCVR;
687 end rcvr_pkg;


688 ------------------------------------------------------------
689 --File : UTILIZATION_CALCULATOR_PKG
690 --Author : Mike Coleman, CAPT, USMC
691 --Project : Channel CAT (Thesis project)
692 --Date : September 1997
693 --Notes : Any personal comment by author will use
694 --        standard comments followed by initials (mgc).
695 --Description : This package is designed provide formatting
696 --              of the results from Channel CAT.  To ease
697 --              interpretation, this function computes the
698 --              maximum sustainable buffer size for the
699 --              current data_period, computes an integer
700 --              solution for percentage of usage at the
701 --              actual time, and sends the utilization to
702 --              the results panel in TAE.  These calculations
703 --              are dynamically based on the user supplied
704 --              data_rate, which is converted to bytes/sec.
705 ------------------------------------------------------------
706 with System;use System;
707 with Text_IO;

708 -- Specification for UTILIZATION_CALCULATOR_PKG
```

```
709 package UTILIZATION_CALCULATOR_PKG is

710     -- Procedure: UTILIZATION_CALCULATOR
711     -- Purpose : Operator which computes the percentage of the max
712     --           channel capacity currently being used.
713     -- Parameters: BUFFER_SIZE : integer value containing the current
714     --                           size of the buffer.
715     --             DATA_PERIOD : integer value which is the period at
716     --                           which data is being sent to channel.
717     --             DATA_RATE : integer value which contain the current
718     --                         speed of the channel.
719     --             START_STOP :  a boolean value used to designate
720     --                           whether the channel should be running or
721     --                           paused (paused refers to being "froze").
722     --             UTILIZATION : integer value sent to the results panel
723     --                           for display as the used percentage.
724    procedure UTILIZATION_CALCULATOR (BUFFER_SIZE : IN INTEGER;
725                                      DATA_PERIOD : IN INTEGER;
726                                      DATA_RATE : IN INTEGER;
727                                      START_STOP : IN BOOLEAN;
728                                      UTILIZATION : OUT INTEGER);

729 end UTILIZATION_CALCULATOR_PKG;



730 -- Body for UTILIZATION_CALCULATOR_PKG

731 package body UTILIZATION_CALCULATOR_PKG is

732    package Util_IO is new Text_IO.Integer_IO(Integer);

733    procedure UTILIZATION_CALCULATOR (BUFFER_SIZE : IN INTEGER;
734                                      DATA_PERIOD : IN INTEGER;
735                                      DATA_RATE : IN INTEGER;
736                                      START_STOP : IN BOOLEAN;
737                                      UTILIZATION : OUT INTEGER) is
738    buffer_maximum : integer := (DATA_RATE * DATA_PERIOD);
739    tex_UTILIZATION : integer;
740    begin
741       UTILIZATION := integer(100.0*float(float(BUFFER_SIZE)/float(buffer_maximum)));
742       tex_UTILIZATION := integer(100.0*float(float(BUFFER_SIZE)/float(buffer_maximum)));
743       Text_IO.Put("Channel utilization is : ");
744       Util_IO.Put(tex_UTILIZATION);
745       Text_IO.Put_Line(" %");
746    end UTILIZATION_CALCULATOR;
747 end UTILIZATION_CALCULATOR_PKG;


748 ------------------------------------------------------------
749 --File : XMTR_PKG
750 --Author : Mike Coleman, CAPT, USMC
751 --Project : Channel CAT (Thesis project)
752 --Date : September 1997
753 --Notes : Any personal comment by author will use
754 --        standard comments followed by initials (mgc).
755 --Description : This package is designed to simulate the
756 --              portion of a TRI-TAC link best described as
```

```
757 --              the transmitter, which places data onto the
758 --              channel.  In ChannelCAT, the xmtr places
759 --              data on the channel by passing a positive
760 --              integer value to channel_buffer, representing
761 --              the byte size of the data.  A state variable,
762 --              period counter is used to allow xmtr to be
763 --              statically scheduled at a period of 1000ms,
764 --              but manage complete firing of the operator
765 --              based on the user supplied data_period.
766 ------------------------------------------------------------
767 with Text_IO;

768 -- Specification for XMTR_PKG

769 package XMTR_PKG is

770     -- Procedure: XMTR
771     -- Purpose : Operator which acts as the "transmitter" in Channel CAT.
772     -- Parameters: DATA_PERIOD : integer value which is the period at
773     --                           which data is to sent to the channel.
774     --             DATA_SIZE : integer value which is the size of the
775     --                         data element being transmitted.
776     --             START_STOP :  a boolean value used to designate
777     --                           whether the channel should be running or
778     --                           paused (paused refers to being "froze").
779     --             BUFFER_MODIFICATION : an integer data type which
780     --                                   can be received from either the
781     --                                   rcvr or xmtr.
782     --             PERIOD_COUNTER : local integer state value used to
783     --                              control complete firing of XMTR to
784     --                              comply with user selected period.
785     procedure XMTR(DATA_PERIOD : IN INTEGER;
786                    DATA_SIZE : IN INTEGER;
787                    START_STOP : IN BOOLEAN;
788                    BUFFER_MODIFICATION : OUT INTEGER;
789                    PERIOD_COUNTER : IN OUT INTEGER );

790 end XMTR_PKG;



791 -- Body for XMT_PKG

792 package body XMTR_PKG is

793     procedure XMTR (DATA_PERIOD : IN INTEGER;
794                     DATA_SIZE : IN INTEGER;
795                     START_STOP : IN BOOLEAN;
796                     BUFFER_MODIFICATION : OUT INTEGER;
797                     PERIOD_COUNTER : IN OUT INTEGER ) is
798     begin
799        -- Increment the period_counter
800        PERIOD_COUNTER := PERIOD_COUNTER + 1;
801        -- Now check the period_counter to see if it matches
802        -- the user supplied DATA_PERIOD.  The period for xmtr
803        -- is 1000ms, so the period_counter will be incremented
804        -- every second, which is quantized increment for user
805        -- supplied data_period values.
806        if PERIOD_COUNTER = DATA_PERIOD then
```

```
807          BUFFER_MODIFICATION := DATA_SIZE;
808          PERIOD_COUNTER := 0;
809       else
810          BUFFER_MODIFICATION := 0;
811       end if;
812    end XMTR;
813 end XMTR_PKG;
```

APPENDIX C. USER INTERFACE CODE

```
 1 ------------------------------------------------------------
 2 --File : USER_INTERFACE_PKG
 3 --Author : Mike Coleman, CAPT, USMC
 4 --Project : Channel CAT (Thesis project)
 5 --Date : September 1997
 6 --Notes : Any personal comment by author will use
 7 --        standard comments followed by initials (mgc).
 8 --Description : This package is designed to implement the
 9 --              user interface for Channel CAT in TAE+.
10 ------------------------------------------------------------
11 --*** TAE+ WorkBench version V5.3 ***
12 ------------------------------------------------------------------------
13 ------------------------------------------------------------------------
14

15 with tae; use tae;
16 with X_Windows;
17 with text_io;

18 -- Specification for USER_INTERFACE_PKG

19 package user_interface_pkg is

20     -- Procedure: USER_INTERFACE
21     -- Purpose : Provide a graphical interface to the Channel CAT
22     --           prototype.  This is a two panel interface, one panel
23     --           dedicated to the input of run-time parameters and the
24     --           second panel used to graphically display results.
25     -- Parameters: UTILIZATION : an integer data type returned from the
26     --                           prototype. Represents the percentage of
27     --                           the maximum channel capacity which is
28     --                           currently being utilized.
29     --             DATA_PERIOD : an integer value sent to the prototype
30     --                           after being set by the user to designate
31     --                           the frequency of data transmission to the
32     --                           channel.
33     --             DATA_SIZE :   an integer value sent to the prototype
34     --                           after being set by the user to designate
35     --                           the size of the data element/object to be
36     --                           sent over the channel.
37     --             START_STOP :  a boolean value sent to the prototype
38     --                           after being set by the user to designate
39     --                           whether the channel should be running or
40     --                           paused (paused refers to being "froze").
41     procedure USER_INTERFACE ( UTILIZATION : IN INTEGER;DATA_PERIOD : OUT
42 INTEGER;DATA_RATE :  OUT INTEGER;DATA_SIZE : OUT INTEGER;START_STOP : OUT BOOLEAN);
43 --(mgc)

44     package taefloat_io is new text_io.float_io (taefloat);
45     procedure initializePanels (file : in string);  -- NOTE: params changed

46     -- BEGIN EVENT_HANDLERs
47     -- Event handler for changes to the DATA_PERIOD parameter
48     procedure params_input_period (info : in tae_wpt.event_context_ptr;period_entry : out
49 INTEGER);
```

```
50      -- Event handler for changes to the DATA_RATE parameter
51      procedure params_input_rate (info : in tae_wpt.event_context_ptr;rate_entry : out
52  INTEGER);

53      -- Event handler for changes to the DATA_SIZE parameter
54      procedure params_input_size (info : in tae_wpt.event_context_ptr;size_entry : out
55  INTEGER);

56      -- Event handler for changes to the START_STOP parameter
57      procedure params_start_button (info : in tae_wpt.event_context_ptr;start_entry : out
58  BOOLEAN);

59      -- END EVENT_HANDLERs

60  end user_interface_pkg;


61  -- Body for USER_INTERFACE_PKG

62  with tae;
63  with Text_IO;

64  package body user_interface_pkg is

65      use tae.tae_misc;

66      theDisplay : X_Windows.Display;
67      Application_Done : boolean := false;
68      user_ptr : tae_wpt.event_context_ptr;
69      params_info : tae_wpt.event_context_ptr;
70      results_info : tae_wpt.event_context_ptr;
71      etype : wpt_eventtype;
72      wptEvent : tae_wpt.wpt_eventptr;

73  -- Declared local variables to prevent reset to zero when
74  -- there is no user input.
75      local_data_period : integer := 1;
76      local_data_rate : integer := 1;
77      local_data_size : integer := 1;
78      local_start_stop : boolean := FALSE;

79  procedure initializePanels (file : in string) is

80
81  use tae.tae_co;
82
83  use tae.tae_misc;

84
85  tmp_info : tae_wpt.event_context_ptr;
86
87      begin

88
89  -- do one Co_New and Co_ReadFile per resource file
90
91  tmp_info := new tae_wpt.event_context;
92
93  Co_New (0, tmp_info.collection);
```

```
 94
 95 -- could pass P_ABORT if you prefer
 96
 97 Co_ReadFile (tmp_info.collection, file, P_CONT);

 98        -- pair of Co_Finds for each panel in this resource file

 99         params_info := new tae_wpt.event_context;
100         params_info.collection := tmp_info.collection;
101         Co_Find (params_info.collection, "params_v", params_info.view);
102         Co_Find (params_info.collection, "params_t", params_info.target);
103
104         results_info := new tae_wpt.event_context;
105         results_info.collection := tmp_info.collection;
106         Co_Find (results_info.collection, "results_v", results_info.view);
107         Co_Find (results_info.collection, "results_t", results_info.target);
108

109        -- Since there can now be MULTIPLE INITIAL PANELS defined from
110        -- within the TAE WorkBench, call Wpt_NewPanel for each panel
111        -- defined to be an initial panel (but not usually all the panels
112        -- which appear in the resource file).

113
114         if params_info.panel_id = NULL_PANEL_ID then
115             tae_wpt.Wpt_NewPanel (theDisplay, params_info.target, params_info.view,
116                X_Windows.Null_Window, params_info, tae_wpt.WPT_PREFERRED,
117                params_info.panel_id);
118         else
119             tae_wpt.Wpt_SetPanelState (
120                params_info.panel_id, tae_wpt.WPT_PREFERRED);
121         end if;

122         if results_info.panel_id = NULL_PANEL_ID then
123             tae_wpt.Wpt_NewPanel (theDisplay, results_info.target, results_info.view,
124                X_Windows.Null_Window, results_info, tae_wpt.WPT_PREFERRED,
125                results_info.panel_id);
126         else
127             tae_wpt.Wpt_SetPanelState (
128                results_info.panel_id, tae_wpt.WPT_PREFERRED);
129         end if;

130      end initializePanels;

131 --
132 --
133 BEGIN EVENT_HANDLERs
134 --
135
136 procedure params_input_period (info : in tae_wpt.event_context_ptr;
137                                period_entry : out integer) is
138     value : array (1..1) of taefloat;
139     count : taeint;
140
141         begin
142         --text_io.put ("Panel params, parm input_period: value = ");
143         tae_vm.Vm_Extract_Count (info.parm_ptr, count);
144         if count <= 0 then
145            --text_io.put_line ("none");
```

```
146            null;
147         else
148            tae_vm.Vm_Extract_RVAL (info.parm_ptr, 1, value(1));
149            --taefloat_io.put (value(1));
150            --text_io.new_line;
151         end if;
152         -- Take the value which was input and send it to the
153         -- prototype via the output parameter
154         period_entry := integer(value(1));
155     end params_input_period;
156
157 procedure params_input_rate (info : in tae_wpt.event_context_ptr;
158                              rate_entry : out integer) is
159     value : array (1..1) of taeint;
160     count : taeint;
161
162         begin
163         --text_io.put ("Panel params, parm input_rate: value = ");
164         tae_vm.Vm_Extract_Count (info.parm_ptr, count);
165         if count <= 0 then
166            --text_io.put_line ("none");
167            null;
168         else
169            tae_vm.Vm_Extract_IVAL (info.parm_ptr, 1, value(1));
170            --text_io.put_line (taeint'image(value(1)));
171         end if;
172         -- Take the value which was input and send it to the
173         -- prototype via the output parameter
174         rate_entry := integer(value(1));
175     end params_input_rate;
176
177 procedure params_input_size (info : in tae_wpt.event_context_ptr;
178                              size_entry : out integer) is
179     value : array (1..1) of taeint;
180     count : taeint;

181     package Util_IO is new Text_IO.Integer_IO(Integer);
182
183         begin
184         --text_io.put ("Panel params, parm input_size: value = ");
185         tae_vm.Vm_Extract_Count (info.parm_ptr, count);
186         if count <= 0 then
187            --text_io.put_line ("none");
188            null;
189         else
190            tae_vm.Vm_Extract_IVAL (info.parm_ptr, 1, value(1));
191            --text_io.put_line (taeint'image(value(1)));
192         end if;
193         -- Take the value which was input and send it to the
194         -- prototype via the output parameter
195         size_entry := integer(value(1));
196     end params_input_size;
197
198 procedure params_start_button (info : in tae_wpt.event_context_ptr;
199                                start_entry : out boolean) is
200     value : array (1..1) of string (1..tae_taeconf.STRINGSIZE);
201     count : taeint;
202
203         begin
```

```
204         --text_io.put ("Panel params, parm start_button: value = ");
205         tae_vm.Vm_Extract_Count (info.parm_ptr, count);
206         if count <= 0 then
207            --text_io.put_line ("none");
208            null;
209         else
210            tae_vm.Vm_Extract_SVAL (info.parm_ptr, 1, value(1));
211            --text_io.put_line (value(1));
212         end if;
213         -- Take the value which was input and send it to the
214         -- prototype via the output parameter
215         -- This requires testing the local value of the output
216         -- parameter to determine the proper boolean change.
217         if local_start_stop = TRUE then -- prototype is running
218            start_entry := FALSE; -- set prototype to stop
219            local_start_stop := FALSE; -- update local variable
220         else if local_start_stop = FALSE then -- prototype is stopped
221               start_entry := TRUE; -- set prototype to run
222               local_start_stop := TRUE; -- update local variable
223            end if;
224         end if;

225      end params_start_button;
226
227
228 --
229 END EVENT_HANDLERs

230 -------------------------------------------------------------------------

231 --
232 -- Main Program
233 --

234 procedure USER_INTERFACE ( UTILIZATION : IN INTEGER;
235                             DATA_PERIOD : OUT INTEGER;
236                             DATA_RATE :  OUT INTEGER;
237                             DATA_SIZE : OUT INTEGER;
238                             START_STOP : OUT BOOLEAN) is

239   begin

240         tae_wpt.Wpt_NextEvent (wptEvent, etype);     -- get next event

241 -- NOTE: This case statement includes STUBs for non-WPT_PARM_EVENT events.

242         case etype is

243             when wpt_eventtype'first .. -1 => null;
244                 -- iterate loop on Wpt_NextEvent error

245 -- TYPICAL CASE: Panel Event (WPT_PARM_EVENT)

246             when tae_wpt.WPT_PARM_EVENT =>
247             -- You can comment out the following "put" call.
248             -- The appropriate EVENT_HANDLER finishes the message.
249                     --text_io.put ( "Event: WPT_PARM_EVENT, " );

250             --      Panel event has occurred.
```

```
251             --      Get parm name and then call appropriate EVENT_HANDLER.
252             --
253             --                          CAUTION:
254             --      DO NOT call Wpt_Extract_Parm_xEvent from any other branch
255             --      of this "case" statement or you'll get "storage_error".
256             --
257                     tae_wpt.Wpt_Extract_Context (wptEvent, user_ptr);
258                     tae_wpt.Wpt_Extract_Parm (wptEvent, user_ptr.parm_name);
259                     tae_wpt.Wpt_Extract_Data (wptEvent, user_ptr.datavm_ptr);
260                     tae_vm.Vm_Find (user_ptr.datavm_ptr, user_ptr.parm_name,
261                                     user_ptr.parm_ptr);
262
263                 -- WPT_PARM_EVENT, BEGIN panel params
264
265                 if tae_wpt."=" (user_ptr, params_info)  then
266                     -- determine appropriate EVENT_HANDLER for this item
267                     if s_equal ("input_period", user_ptr.parm_name) then
268                         params_input_period (user_ptr,local_data_period);
269                     elsif s_equal ("input_rate", user_ptr.parm_name) then
270                         params_input_rate (user_ptr,local_data_rate);
271                     elsif s_equal ("input_size", user_ptr.parm_name) then
272                         params_input_size (user_ptr,local_data_size);
273                     elsif s_equal ("start_button", user_ptr.parm_name) then
274                         params_start_button (user_ptr,local_start_stop);
275                     end if;     -- END panel params
276
277                 -- WPT_PARM_EVENT, BEGIN panel results
278                 else
279                     text_io.put_line ("unexpected event from wpt!");
280                     --exit; -- or raise an exception, but compiler warns if no exit
281                 end if;

282             when tae_wpt.WPT_FILE_EVENT =>
283                         text_io.put_line ("STUB: Event WPT_FILE_EVENT");

284                         -- Use Wpt_AddEvent and Wpt_RemoveEvent and
285                         -- Wpt_Extract_EventSource and Wpt_Extract_EventMask

286             when tae_wpt.WPT_TIMEOUT_EVENT =>
287                         --text_io.put_line ("STUB: Event WPT_TIMEOUT_EVENT");
288                         null;
289                         -- Use Wpt_SetTimeOut for this

290             when tae_wpt.WPT_TIMER_EVENT =>
291                         text_io.put_line ("STUB: Event WPT_TIMER_EVENT");

292                         -- Use Wpt_AddTimer and Wpt_RemoveTimer and
293                         -- Wpt_Extract_TimerId, Wpt_ExtractTimerRepeat,
294                         -- and Wpt_Extract_TimerInterval

295 -- LEAST LIKELY cases follow:

296             when tae_wpt.WPT_WINDOW_EVENT => null ;


297                         -- WPT_WINDOW_EVENT can be caused by user acknowledgement
298                         -- of a Wpt_PanelMessage or windows which you
299                         -- directly create with X (not TAE panels).
300                         -- You MIGHT want to use Wpt_Extract_xEvent_Type here.
```

```
301                         --
302                         -- DO NOT use Wpt_Extract_Parm_xEvent since this is not
303                         -- a WPT_PARM_EVENT; you'll get a "storage error".

304             when tae_wpt.WPT_HELP_EVENT =>          -- OR null ;
305                         text_io.put("ERROR: WPT_HELP_EVENT: ");
306                         text_io.put_line("should never see; reserved for TAE use");

307             when tae_wpt.WPT_INTERRUPT_EVENT =>     -- OR null ;
308                         text_io.put("ERROR: WPT_INTERRUPT_EVENT: ");
309                         text_io.put_line("should never see; reserved for TAE use");

310             when OTHERS =>
311                         text_io.put ("FATAL ERROR: Unknown Wpt_NextEvent Event Type: ");
312                         text_io.put (wpt_eventtype'image(etype) ) ;
313                         text_io.put_line (" ... Forcing exit.");
314                         --exit;  -- or raise an exception

315         end case;    -- NOTE: Do not add statements between here and "end loop EVENT_LOOP"

316 -- This single line connects the input integer value, UTILIZATION, to
317 -- the results panel and sends the value to "graph" as a real number,
318 -- which is a required conversion due to the expected data type of a
319 -- stripchart generated using TAE.
320 TAE_WPT.WPT_SETREAL(results_info.panel_id, "graph", TAEFLOAT(UTILIZATION));

321 -- Update all out parameters using local variables
322 data_rate := local_data_rate;
323 data_period := local_data_period;
324 data_size := local_data_size;
325 start_stop := local_start_stop;

326 end USER_INTERFACE;

327 begin


328 Text_IO.Put_Line("Channel CAT (Capacity Analysis Tool)");
329 Text_IO.Put_Line("Developed by Capt. Mike Coleman, USMC");
330 Text_IO.Put_Line("-------------------------------------");

331     f_force_lower (FALSE);    -- permit upper/lowercase file names
332     tae_wpt.Wpt_Init ("",theDisplay);

333     tae_wpt.Wpt_NewEvent (wptEvent);
334     initializePanels ("channel_cat.res");   -- single call
335     tae_wpt.wpt_settimeout(1);


336 end user_interface_pkg;
```

APPENDIX D. TESTING/VALIDATION CRITERIA AND RESULTS

Channel_CAT_Testing/Validation_Criteria

```
Validation set #1 (Boundary checks):
====================================
Test 1 (lower boundary):
------------------------
   data size : 1 byte
   data period : 1 second
   data rate : 1 byte/second
   Expected results : 100 % use
Test 2 (upper boundary):
------------------------
   data size : 4000 bytes
   data period : 1 second
   data rate : 4000 bytes/second
   Expected results : 100 % use
Test 3 (max capacity @ half speed):
-----------------------------------
   data size : 4000 bytes
   data period : 2 seconds
   data rate : 2000 bytes/second
   Expected results : 100 % use
```

```
Validation set #2 (50% max capacity):
=====================================
Test #1 (low rate/size):
------------------------
   data size : 1 byte
   data period : 2 seconds
   data rate : 1 byte/second
   Expected results : 50 %
Test #2 (high rate/size):
-------------------------
   data size : 4000 bytes
   data period : 2 seconds
   data rate : 4000 bytes/second
   Expected results : 50 %
Test #3 (high rate/mid size):
-------------------------
   data size : 2000 bytes
   data period : 1 second
   data rate : 4000 bytes/second
   Expected results : 50 %
Test 3.b (mid rate/size):
-------------------------
   data size : 2000 bytes
   data period : 2 seconds
   data rate : 2000 bytes/second
   Expected results : 50 %
```

```
Validation set #3 (parameter impact):
=====================================
Control parameters (reference):
-------------------------------
   data size : 1000 bytes
   data period : 10 seconds
   data rate : 100 bytes/second
   Expected results : 100 %
Test #1 (excessive data size):
------------------------------
   data size : 1001 bytes
   data period : 10 seconds
   data rate : 100 bytes/second
   Expected results : >100 %
Test #2 (data transmitted too often):
-------------------------------------
   data size : 1000 bytes
   data period : 9 seconds
   data rate : 100 bytes/second
   Expected results : >100 %
Test #3 (data rate is too slow):
--------------------------------
   data size : 1000 bytes
   data period : 10 seconds
   data rate : 99 bytes/second
   Expected results : >100 %
Test #4 (data size not max):
----------------------------
   data size : 999 bytes
   data period : 10 seconds
   data rate : 100 bytes/second
   Expected results : <100 %
Test #5 (period not max):
-------------------------
   data size : 1000 bytes
   data period : 11 seconds
   data rate : 100 bytes/second
   Expected results : <100 %
Test #6 (rate faster than needed):
----------------------------------
   data size : 1000 bytes
   data period : 10 seconds
   data rate : 101 bytes/second
   Expected results : <100 %
```

Channel CAT Testing/Validation Results

```
Validation set #1 (Boundary checks):
====================================
Test 1 (lower boundary):
------------------------
   data size : 1 byte
   data period : 1 second
   data rate : 1 byte/second
   Expected results : 100 % use
   Graphical results : peak at 100% (red) and
                       depletes to 0.
   Notes & comments : PASS w/ no restrictions.
Test 2 (upper boundary):
------------------------
   data size : 4000 bytes
   data period : 1 second
   data rate : 4000 bytes/second
   Expected results : 100 % use
   Graphical results : peak at 100% (red) and
                       depletes to 0.
   Notes & comments : PASS w/ no restrictions.
Test 3 (max capacity @ half speed):
-----------------------------------
   data size : 4000 bytes
   data period : 2 seconds
   data rate : 2000 bytes/second
   Expected results : 100 % use
   Graphical results :peak at 100% (red) and
                      depleted to 0, but at
                      half the rate compared to
                      above.
   Notes & comments : PASS w/ no restrictions.
```

```
Validation set #2 (50% max capacity):
=====================================
Test #1 (low rate/size):
------------------------
   data size : 1 byte
   data period : 2 seconds
   data rate : 1 byte/second
   Expected results : 50 %
   Graphical results : peak at 50% and then
                       depleted to 0.
   Notes & comments : PASS w/ no restrictions.
Test #2 (high rate/size):
-------------------------
   data size : 4000 bytes
   data period : 2 seconds
   data rate : 4000 bytes/second
   Expected results : 50 %
   Graphical results : peak at 50% and then
                       depleted to 0.
   Notes & comments : PASS w/ no restrictions.
Test #3 (high rate/mid size):
-------------------------
   data size : 2000 bytes
   data period : 1 second
   data rate : 4000 bytes/second
   Expected results : 50 %
   Graphical results : peak at 50% and then
                       depleted to 0.
   Notes & comments : PASS w/ no restrictions.
Test 3.b (mid rate/size):
-------------------------
   data size : 2000 bytes
   data period : 2 seconds
   data rate : 2000 bytes/second
   Expected results : 50 %
   Graphical results : peak at 50% and then
                       depleted to 0.
   Notes & comments : PASS w/ no restrictions.
```

```
Validation set #3 (parameter impact):
=====================================
Control parameters (reference):
-------------------------------
   data size : 1000 bytes
   data period : 10 seconds
   data rate : 100 bytes/second
   Expected results : 100 %
   Graphical results : peak at 100% (red), and
                       incrementally depleted
                       by 5% down to 0.
   Notes & comments : PASS w/ no restrictions.
                      Control conditions valid.
Test #1 (excessive data size):
------------------------------
   data size : 1001 bytes
   data period : 10 seconds
   data rate : 100 bytes/second
   Expected results : >100 %
   Graphical results : The impact of increasing
                       the data size took 5
                       transmission periods before
                       an increasing usage was seen.
                       Increases were 1% amounts.
   Notes & comments : delay in utilization being
                      displayed is due to rounding
                      of display values.  Actual tool
                      variables held accurate values
                      and maintain fidelity.
                       PASS with above restriction noted.
Test #2 (data transmitted too often):
-------------------------------------
   data size : 1000 bytes
   data period : 9 seconds
   data rate : 100 bytes/second
   Expected results : >100 %
   Graphical results : Changes from control conditions
                       were immediately visible, and the
                       changes to utilization's max value
                       were much larger than before (11%).
   Notes & comments : The impact of changing the period
                      had a fairly dramatic effect.
                       PASS w/ no restrictions.
Test #3 (data rate is too slow):
--------------------------------
   data size : 1000 bytes
   data period : 10 seconds
   data rate : 99 bytes/second
   Expected results : >100 %
   Graphical results : Utilization value was immediately
                       affected, causing a slow, but steady
                       increase in peak utilization by 1%
                       each transmission.
   Notes & comments : Impact of decreasing the rate was
                      quickly noticable, but small.
                       PASS w/ no restrictions.
Test #4 (data size not max):
----------------------------
   data size : 999 bytes
```

```
   data period : 10 seconds
   data rate : 100 bytes/second
   Expected results : <100 %
   Graphical results : No changes from control conditions
                       noted (5 minutes run time).  Mid-test
                       changes to data size finally yielded a
                       change to displayed utilization at 994.
   Notes & comments : Necessity of changing data size to smaller
                      value is also explained by integer rounding
                      being done with UTILIZATION_CALCULATOR.
                       PASS with above restriction noted.
Test #5 (period not max):
-------------------------
   data size : 1000 bytes
   data period : 11 seconds
   data rate : 100 bytes/second
   Expected results : <100 %
   Graphical results : Change to utilization was immediate.
   Notes & comments : PASS w/ no restrictions.
Test #6 (rate faster than needed):
----------------------------------
   data size : 1000 bytes
   data period : 10 seconds
   data rate : 101 bytes/second
   Expected results : <100 %
   Graphical results : Change to utilization was immediate.
   Notes & comments : PASS w/ no restrictions.
```

INITIAL DISTRIBUTION LIST

Number of Copies

1. Defense Technical Information Center 2
   8725 John J. Kingman Road., Ste 0944
   Ft. Belvoir, VA. 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Director, Training and Education 1
   MCCDC, Code C46
   1019 Elliot Rd.
   Quantico, VA 22134-5027

4. Director, Marine Corps Research Center 2
   MCCDC, Code C40RC
   2040 Broadway Street
   Quantico, VA 22134-5107

5. Director, Studies and Analysis Division 1
   MCCDC, Code C45
   3300 Russell Road
   Quantico, VA 22134-5130

6. Marine Corps Representative 1
   Naval Postgraduate School
   Code 037, Bldg. 234, HA-220
   699 Dyer Road
   Monterey, CA 93940

7. Marine Corps Tactical Systems Support Activity 1
   Technical Advisory Branch
   Attn: Maj. J.C. Cummiskey
   Box 555171
   Camp Pendleton, CA 92055-5080

8. Dr. Luqi, Code CS/Lq.....5
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943-5002

9. CDR Michael J. Holden USN, Code CS/Hm.....1
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943-5002

10. Dr. Ted Lewis, Code CS/Le.....1
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943-5002

11. ECJ6-NP.....1
HQUSEUCOM
Unit 30400 Box 1000
APO, AE 09128

12. Capt. Michael G. Coleman USMC.....2
572 Holloway Road
Annapolis, MD
21402

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

DUDLEY KNOX LIBRARY
3 2768 00338551 9